scientific reports

Check for updates

OPEN

Fast learning without synaptic plasticity in spiking neural networks

Anand Subramoney[1,2], Guillaume Bellec[1,3], Franz Scherr[1], Robert Legenstein[1] & Wolfgang Maass[1]✉

Spiking neural networks are of high current interest, both from the perspective of modelling neural networks of the brain and for porting their fast learning capability and energy efficiency into neuromorphic hardware. But so far we have not been able to reproduce fast learning capabilities of the brain in spiking neural networks. Biological data suggest that a synergy of synaptic plasticity on a slow time scale with network dynamics on a faster time scale is responsible for fast learning capabilities of the brain. We show here that a suitable orchestration of this synergy between synaptic plasticity and network dynamics does in fact reproduce fast learning capabilities of generic recurrent networks of spiking neurons. This points to the important role of recurrent connections in spiking networks, since these are necessary for enabling salient network dynamics. We show more specifically that the proposed synergy enables synaptic weights to encode more general information such as priors and task structures, since moment-to-moment processing of new information can be delegated to the network dynamics.

Modelling and theoretical investigation of learning capabilities of models for neural networks of the brain, in particular of networks of spiking neurons, has focused on learning via synaptic plasticity, such as spike-timing-dependent plasticity (STDP). But experimental data suggest that synaptic plasticity does not capture all learning capabilities of neural networks in the brain.

Brains can learn very fast, even in a single trial[1]. In contrast, experimentally grounded rules for synaptic plasticity such as STDP require numerous repetitions of a trial[2], hence this plasticity rule is not likely to be the only mechanisms for fast learning capabilities of brains. A number of other experimental data suggest that brains use, in addition to or instead of synaptic plasticity, the dynamics of network states to store new information[3–5]. It has already been demonstrated that a particular type of artificial neural network, networks of Long Short-Term memory (LSTM) units[6–9] can also accomplish this. However this result provides little information about fast learning capabilities of networks of spiking neurons since their dynamics is quite different. In particular, LSTM units employ registers, similar to digital computers, for rapidly storing information for an indefinite amount of time, which LIF neurons cannot do.

However recently it has been shown that a substantial fraction of the functional capability of networks of LSTM units can be reproduced by networks of spiking neurons, provided that they also contain neurons with spike frequency adaptation (SFA)[10,11]. SFA means that a neuron increases its firing threshold after firing. SFA has already been implicated for quite some while in cellular short-term memory[12,13] and other important features of brain networks[14]. We show that neurons with SFA endow networks of spiking neurons with the capability to learn very fast, even without synaptic plasticity. We focus on two characteristic aspects of the resulting new learning theory for networks of spiking neurons:

1. Synaptic weights are able to encode priors for learning, in particular priors that enable fast learning and generalization from few examples by exploiting common structural aspects of related learning tasks[15].
2. Synaptic weights are able to encode instructions for controlling fast learning processes through the network dynamics with fixed weights. This perspective enables brains to employ a much larger and functionally more powerful repertoire of tow-tiered learning schemes.

[1]Institute for Theoretical Computer Science, Graz University of Technology, Graz, Austria. [2]Department of Computer Science, Royal Holloway University of London, Egham, UK. [3]Laboratory of Computational Neuroscience, Ecole Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland. ✉email: maass@igi.tugraz.at

We demonstrate each of these two principles separately in two illustrative tasks (see Figs. 1 and 4) and together in applications to standard motor control and navigation tasks that require self-supervised learning and reinforcement learning (Figs. 2 and 3).

In line with Refs.[8,9,16,17] we focus on a setting where synaptic weights are tuned on a large time-scale that conceptually reflects evolutionary and developmental processes as well as prior learning. We show that this setup can also be used to elucidate fast learning capabilities of spiking neural networks, i.e. of biologically more realistic models for neural networks of the brain.

Results

We consider recurrent networks of spiking neurons (RSNNs) that contain, besides standard leaky integrate-and-fire (LIF) neurons, also a random subset of neurons with spike-frequency adaptation (SFA)[11]. SFA is a biologically motivated mechanism based on[18] for short-term memory[12,13] that can be efficiently simulated and allows parameter optimization in a neural network model through gradient descent. It has been previously demonstrated that SFA is necessary for spiking neural networks to achieve functional parity with artificial neural networks[11]. Experimental data from the Allen Institute[19] show that a substantial fraction of excitatory neurons of the neocortex, ranging from 20% in mouse visual cortex to 40% in the human frontal lobe, exhibit SFA. Therefore, we endow only a subset of spiking neurons with SFA in line with this data (Also see Fig. 8 in Salaj et al.[11] for a plot of the distribution).

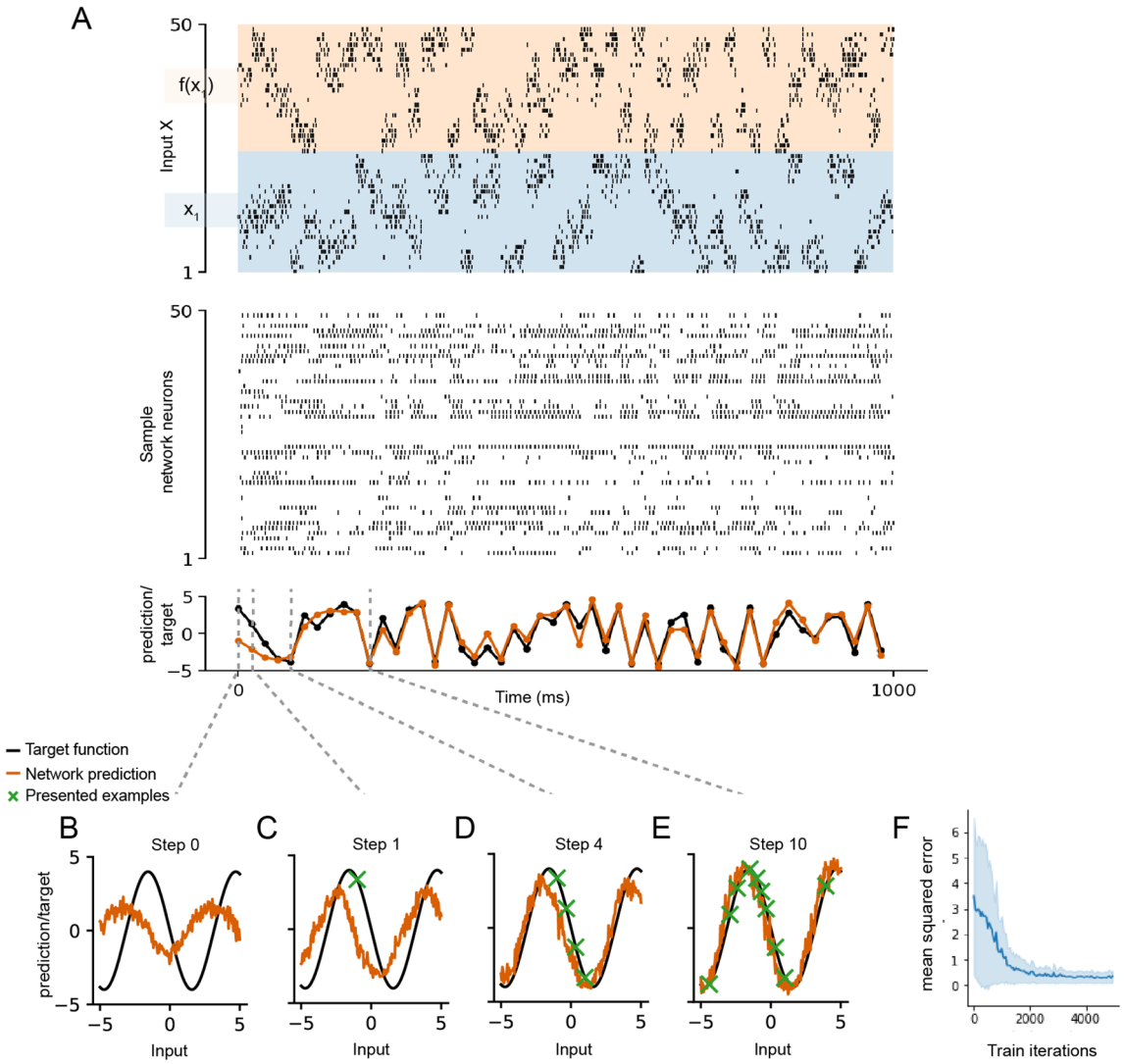

Figure 1. Encoding structural priors for learning. (**A**) Sample spike raster of neurons in the RSNN during an episode of learning to predict a previously unseen sinusoidal curve. (**B**–**E**) Snapshots of the internal model of the network at different inner loop steps, illustration of the prior knowledge acquired by the RSNN containing LIF neurons with SFA through L2L for the family $\mathscr{F}$ of all sinus functions with different phases and amplitudes, but a fixed frequency. Orange curves show the effective internal model of the RSNN at different stages of learning from examples of the target function (marked by green crosses). (**F**) Learning curve showing the loss minimized over the outer loop training.

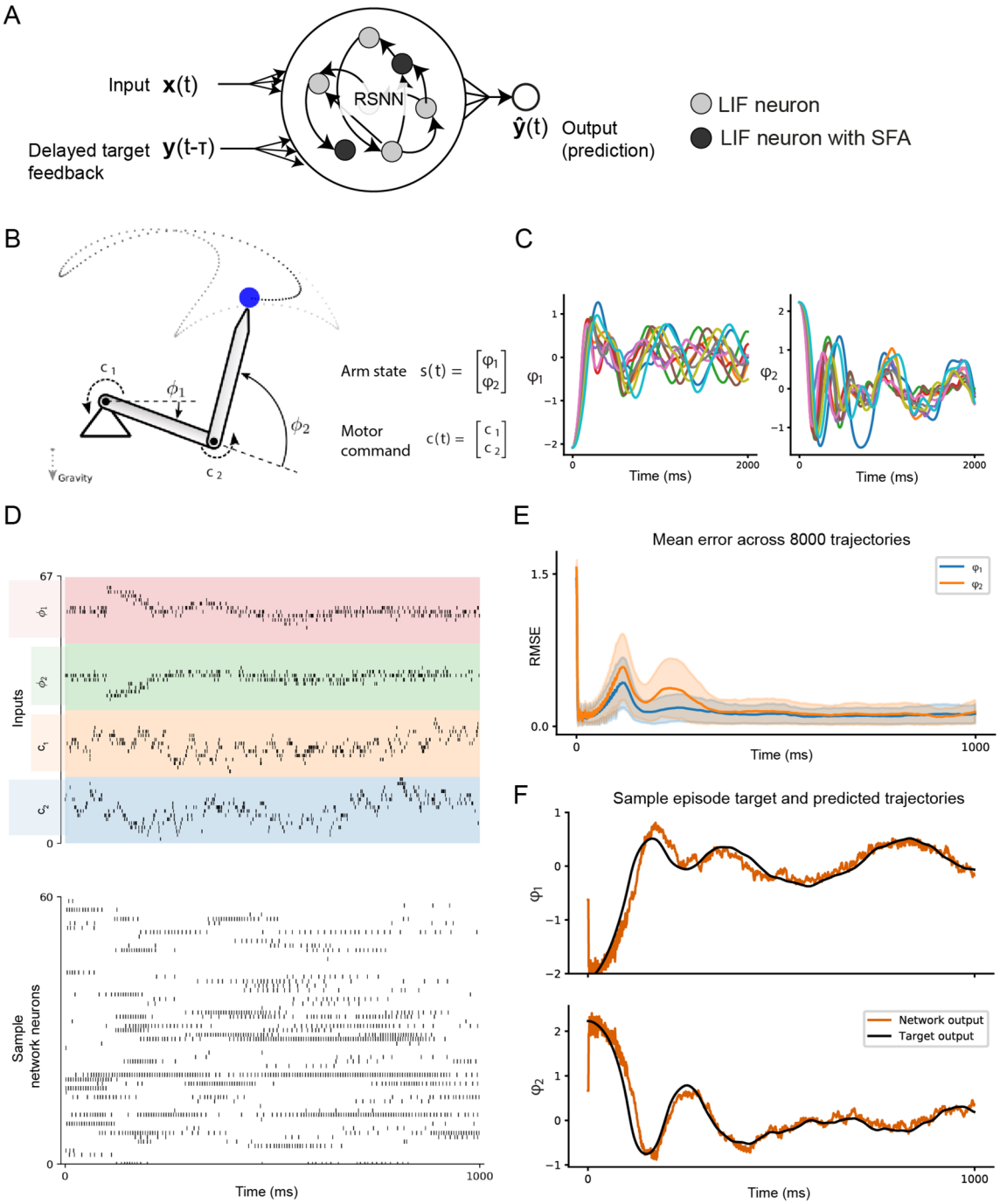

Figure 2. One-shot adaptation of a forward model for an arm (**A**) The network setup for this and Sect. "Priors encoded in synaptic weights can significantly speed up learning" consists of a generic recurrent network of spiking neurons some of which exhibit SFA. The network receives, in addition to the input $x(t)$, a delayed target feedback signal $y(t-\tau)$, and produces the output $\hat{y}(t)$ (where $y(t)$ is the actual target). (**B**) Illustration of the two-link arm model with states given by the angle of the links, and the motor command applied on both the joints. (**C**) Sample trajectories generated for the same torque sequence by arms with different masses and lengths of the limbs. (**D**) Sample spike raster of neurons in the RSNN during an episode. The inputs to the network are, from top to bottom, the $\tau = 100$ ms delayed states $\phi_1(t-\tau)$ and $\phi_2(t-\tau)$ given as feedback; and the motor commands $c_1(t)$ and $c_2(t)$. (**E**) Root mean-squared error over all test episodes during the 1 second of inner loop learning. (**F**) Target trajectories and network prediction for one sample test episode for an arm with new link lengths and masses.

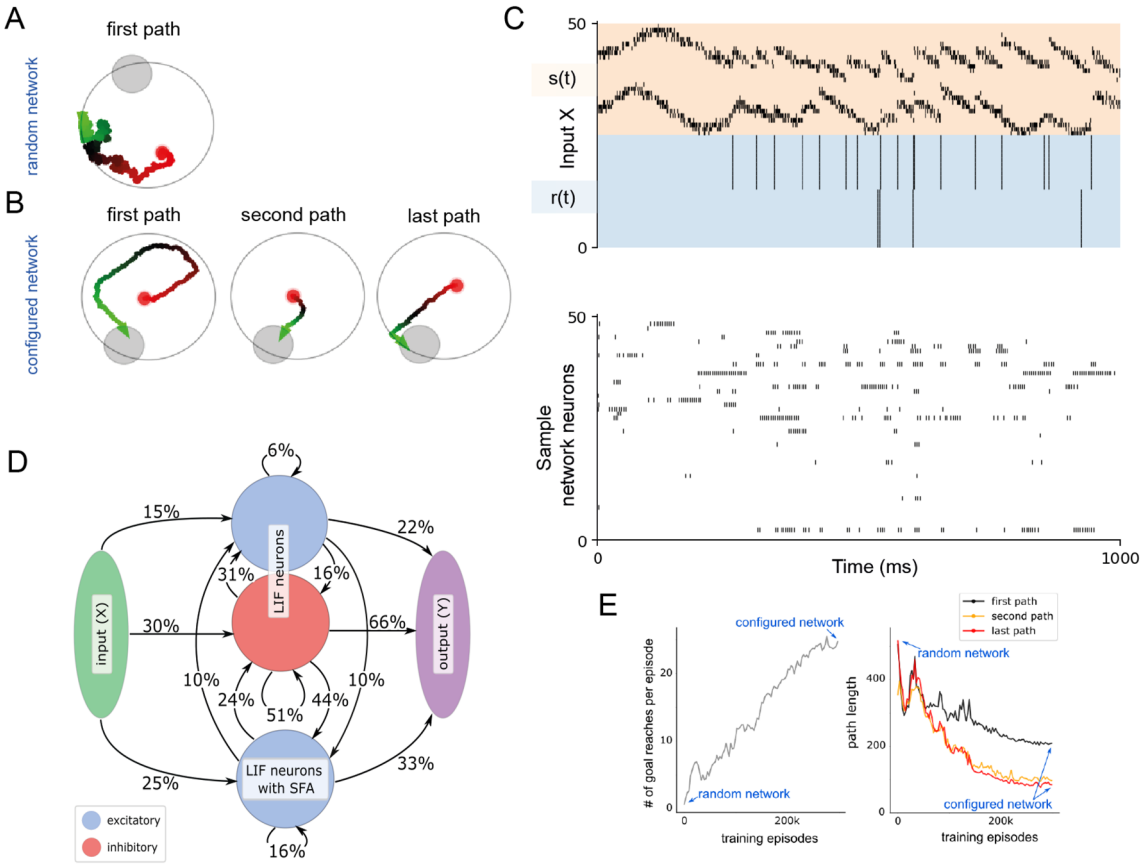

Figure 3. Reward-based learning of an RSNN without synaptic plasticity. (**A**,**B**) Samples of navigation paths produced by the RSNN before and after this training. Before training, the agent performs a random walk (**A**). In this example it does not find the goal within the limited episode duration. After tuning the synaptic weights of the RSNN in the outer loop of L2L (**B**), the RSNN 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 remained the same throughout an episode. Note that all synaptic weights of the RSNNs remained fixed during an episode. (**C**) The network dynamics that produced the behavior shown in (**B**). (**D**) The architecture of the RSNN, consisting of excitatory and inhibitory neurons, with 20% connectivity obeying Dale's law. Synaptic connectivity was optimized within these constraints in the outer loop of L2L. (**E**) Learning curves showing the variation of the number of goals reached and path length through the course of outer loop training.

In addition to the input, the network receives either a cue (in the experiment in Sect. "New learning capabilities of recurrent networks of spiking neurons") or a feedback signal (in all other experiments). We set all the synaptic weights of the network through an optimization process in the learning to learn paradigm to solve the tasks described. See "Methods" Sect. "Details of the learning to learn setup" for more details.

Our analysis builds on a key insight from neuroscience and cognitive science: Fast learning capability of brains is supported by the fact that brains do not learn a new task starting in a tabula-rasa state. Rather, they build on neural circuits, learning skills, and prior knowledge that have been formed throughout evolution, development, and prior learning experiences[9,16,17]. The capabilities of this prior shaping of neural circuits can be analyzed with the help of the formal learning-to-learn (L2L) model from[6–8] as described in "Methods" Sect. Details of the learning to learn setup.

Priors encoded in synaptic weights can significantly speed up learning

To demonstrate that synaptic weights can also be used to encode innate or previously learnt priors that can enable and/or speed up learning of complex tasks (point 1 of the Introduction), we will use a simple task where the RSNN has to learn a mapping f from input values x to output values y from example pairs $\langle x, y\rangle$ with $y = f(x)$. Here, each task C corresponds to a mapping f. This learning task requires generalization from mappings f' that occurred during training to mappings f that did not occur during training. Obviously, this generalization is impossible if the learner has no prior knowledge about the function f that is to be learnt. Artificial neural networks with continuous activation functions implicitly use a prior that the target function f is smooth. But SNNs do not automatically apply a smoothness prior, since they can just as well represent discontinuous input-output mappings. We wondered whether the weights of a RSNN could encode a smoothness prior, and possibly further structural properties of potential target functions f.

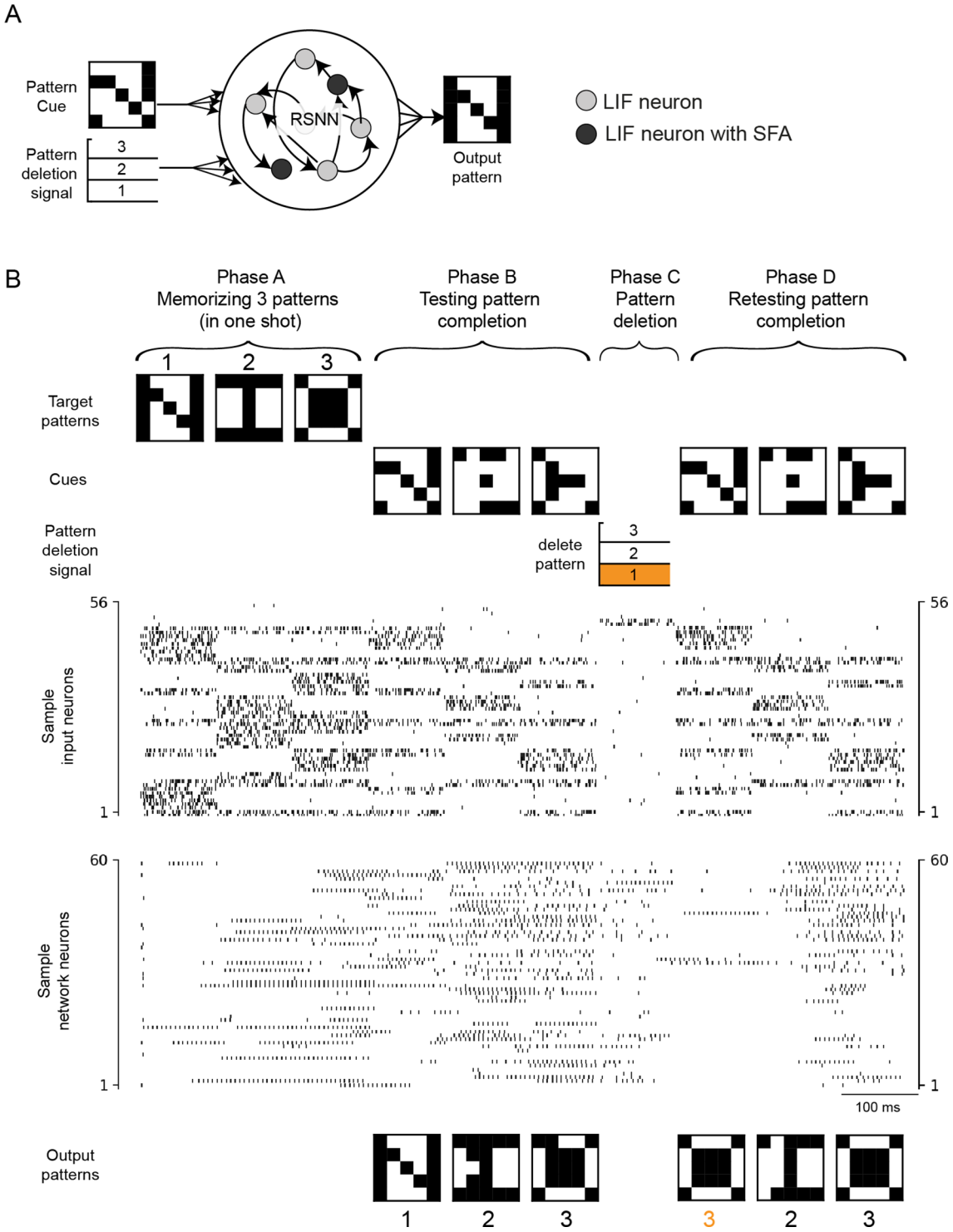

Figure 4. Example of new neural network learning capability that arises when synaptic weights are enabled to store details of the learning strategy. (**A**) The neural network is a generic recurrent network of spiking neurons, some of which exhibit SFA. The network was provided cue patterns and pattern deletion signals as input, and produced the appropriate completed pattern as outputs. (**B**) Demonstration of the capability of the network to learn 3 new attractors in one shot so that they can be used for input completion, and also to delete one of the attractors (here attractor 1) in one shot.

We focused on the specific case where it is a priori known that the target function f is a sinus function, but with unknown phase and amplitude. In each inner loop episode, the RSNN received a sequence of inputs x^k from some mapping f, each encoded through the population activity of 100 spiking neurons for 20 ms. In addition to

x^k, it received the target output $y^{k-1} = f(x^{k-1})$ for the previously presented input (see Methods). In this way, the network received a delayed feedback about its desired output which it could use to adapt its behavior in accordance with its internal prior on the family of functions f. The weights of the RSNN were kept fixed in the inner loop. The network had to predict the target $y^k = f(x^k)$ at each time step k, and was trained in the outer loop using backpropagation-through-time (BPTT) to do so in batches of episodes with a different mappings/tasks. During testing, previously unseen mappings were used.

Figure 1B–E demonstrates that this prior information can in fact be encoded in the synaptic weights of the RSNN, which are determined in the outer loop of L2L. We applied a simple trick for visualization to make the prior or internal model of the RSNN at any moment in time visible: see the orange lines in Fig. 1B–E. These orange lines show for any potential input value x (in the domain of f) the output y which the RSNN would give if this x would occur as the next network input (in a hypothetical experiment, that has no effect on the next steps of the (inner loop) learning process for learning the target function f). More precisely, the network state is not allowed to change when these test inputs x are shown.

Figure 1B shows the prior or internal model that is engraved into the RSNN through its synaptic weights before it has received any training example $\langle x, y \rangle$. One clearly sees that this internal model is in fact a sinus function. In addition, this internal model already reflects the frequency of the target sinus function f, since this is the same for all potential target functions that were considered in the outer loop of L2L. The subsequent panels, Fig. 1C–E, show that the internal model is updated when some actual training examples — indicated by green crosses — are received by the RSNN. One sees from Fig. 1C that a single training example brings the internal model already quite close to the function f from which the training examples are generated. Figure 1D shows that the prior of the RSNN has such a strong impact that even when it receives 4 training examples from f that happen to lie approximately on a straight line, its internal model (i.e, posterior) still has the form of a sinus function, rather than a straight line. Figure 1E shows the internal model once the network has received sufficient number of points to fully predict the sinus function. The normal temporal progression of the experiment at each step is shown in Fig. 1A, where the network state progresses normally after each example (that were marked by green crosses in Fig. 1B–E) is presented to the network. The total test MSE was 0.1968 ± 0.1469 over 5 runs and the linear baseline was 4.0340.

Fast adaptation of motor predictions

The brain is able to adapt its motor control commands very fast, sometimes even in a single trial[5,20]. It is unlikely that synaptic plasticity can accomplish that[3], and an alternative model has been missing. We show that one-shot adaptation of motor prediction can be achieved if synaptic plasticity in the outer loop of L2L is complemented by the capability to transiently store salient information in the network state. We demonstrate this for the case of a forward model for an arm. The brain needs such forward models to plan movements, and also to take corrective actions if needed[21,22]. Visual and proprioceptive feedback provide essential feedback for that[23].

We address the question of how the brain can quickly adapt its motor predictions for arm movements when kinematic or dynamic properties of the arm change. For example, carrying a load changes the distribution of masses over the arm, and using a tool in the hand changes its effective length. And yet, neural networks of the brain can quickly correct for these changes — without requiring multiple rounds of trial and error. Our goal was to produce a model of how RSNNs can achieve this without synaptic plasticity.

Here, we consider the case of a two-link arm as illustrated in Fig. 2B. The tip of the arm is moved by applying torques to each of the two joints. Both of its limbs are also subject to gravity. The task was to predict the angles of the two joints. But the masses and lengths of the two limbs were different in every episode, leading to very different trajectories even when the same torques were applied (Fig. 2C). The RSNN received as input the control torques applied to the arm model encoded through the population activity of 100 spiking neurons (see Fig. 2D and Methods). No direct information about the masses and lengths of the limbs were provided to the model, only the true angles of the limbs were given as feedback to the network with a delay of 100 ms (this feedback was set to 0 for the first 100 ms). The RSNN was trained using BPTT in the outer loop to minimize mean-squared error between predictions and targets, while the weights of the network were kept fixed in the inner loop. Nevertheless the RSNN was able to adapt its predictions for a new arm within about 600 ms while moving it for the first time (Fig. 2E,F). This is substantially faster than previous models for adaptation of a forward model based on synaptic plasticity[24]. Overall, the network with SFA achieved a root mean squared error of 0.0529 m. Essential for this fast adaptation was that the RSNN model included neurons with SFA, and that its synaptic weights were trained in the outer loop of L2L to enable this very fast adaptation (see Methods for details).

Spiking neural networks can learn extremely fast from rewards — without engaging synaptic plasticity

We now demonstrate the ability of synaptic weights to encode innate or previously learnt priors that can enable and/or speed up learning of complex reinforcement learning tasks. For this, we use variations of the well-known Morris water-maze task[25,26] to define the range $\mathscr{F}$ of learning tasks for L2L. Here the subject has to learn to find a target in a 2D arena, and to navigate subsequently to this target from random positions in the arena (Fig. 3A,B).

The task consists of episodes that each last 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 goal reaches within an episode. The Morris water-maze task is related to one of the more challenging demos of Wang et al.[8] and Duan et al.[7] in applying L2L to networks of LSTM units. But it had remained open whether this learning paradigm can also be applied to biologically more realistic neural network models. For added biological plausibility, we investigate

here to what extent a sparsely connected network of excitatory and inhibitory neurons that observes Dale' law can learn to solve the Morris water-maze task within a few trials. To test this, we not only train the weights, but also the connectivity of the network using DEEP R[27].

We are addressing here at the same time point 2 of the Introduction: Can synaptic weights encode common structure in this family of task so that the network can use this abstract knowledge for more efficient learning? Concretely, there are two pieces of abstract knowledge in the family of water-maze tasks: The fact that goals are always on the border of the arena, and the fact that the goal position is constant within each episode. Note that we did not allow synaptic plasticity to take place during the short duration of a testing episode, only in the outer loop of L2L.

Since RSNNs 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 (see orange segment in the top row of Fig. 3C). Note that the lack of visual input already makes it 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 (blue segment of the upper row of Fig. 3C). We used the outer loop of L2L to configure the network to solve this task as fast as possible — see Methods for details of the optimization process used to configure the network. In this task the RSNN had 400 recurrent units (200 excitatory and 80 inhibitory standard LIF neurons, and 120 excitatory neurons with SFA) and a synaptic connectivity of 20%. Allowing the network to rewire itself during the outer loop of L2L substantially improved the performance. The resulting network diagram and spike raster is shown in Fig. 3C. The network achieved a average accumulated reward of 26.76 ± 6.95 over 10 runs.

The first path in Fig. 3B shows that the RSNN is able to make use of the fact that the goal is located on the border of the maze. The second and last paths show that the RSNN also makes use of the abstract knowledge that the goal position remains fixed during an episode. Fig. 3C exhibits sample spike trains from excitatory and inhibitory LIF neurons without SFA, and of excitatory LIF neurons with SFA.

Altogether this demo shows that RSNN are able to learn very fast from rewards, without engaging synaptic plasticity. Furthermore it shows that synaptic weights of SNNs can encode abstract knowledge which makes learning of a behaviour substantially more efficient.

New learning capabilities of recurrent networks of spiking neurons

Here, we want to demonstrate point 2 of the Introduction, the substantially enlarged range of learning strategies that become available if one integrates dynamic network states into the learning process. We demonstrate this, in a limited way, on some of the arguably most important learning goals for recurrent neural networks: learning an attractor, using a learnt attractor for input completion, and deleting an attractor for pattern completion. The first two learning goals can be achieved through Hebbian learning rules in suitable artificial neural networks such as Hopfield networks[28]. However the learning of a new attractor typically requires a substantial number of trials, whereas the brain is able to learn a new rule or prototype for image classification in one or very few trials. Deleting an attractor for pattern completion corresponds to learning that a specific rule or prototype is no longer valid. This can also be accomplished by the human brain in one or few trials, but it is difficult to achieve through training of any type of recurrent neural network. But importantly, none of the three mentioned learning goals have been demonstrated for more realistic models of neural networks such as recurrent networks of spiking neurons. We demonstrate (Fig. 4), in a limited setting with fixed ordering of inputs, that they can achieve all three learning goals very fast, even in a single trial, if one takes into account that synaptic weights can encode a much wider repertoire of learning methods than those that are accessible through local rules for synaptic plasticity such as Hebbian rules or STDP. Due to the limitations in generalisation ability achievable through training recurrent neural networks with backpropagation through time, the network is only able to handle the phases in the order it is trained on.

The recurrent network we use consisted of 300 spiking neurons, half of which exhibited SFA, was trained in the outer loop of L2L to be able to memorize any arbitrary three prototype patterns instantaneously in phase A. The patterns were randomly generated 25-bit patterns, and network performance was evaluated for patterns that did not occur during training in the outer loop. Then in phase B it could use these stored prototype patterns for completing partial network inputs. The network also was able to delete any of the three pattern prototypes (here pattern 1) in phase C, and to continue pattern completion with the remaining two pattern prototypes 2 and 3 (phase D). Note that the same partial network input or cue that lead in phase B to pattern (attractor) 1 is now completed in phase D to the next best prototype pattern 3 (with closest hamming distance).

The network was able to perform this four phase task for arbitrary prototype patterns consisting of 25 bits, achieving for new patterns a bitwise completion accuracy of 97.34% in phase B and 77.52% in phase D (for an average of 87.43% in both phases). See "Methods" for full details.

Discussion

We have revisited the roles of synaptic plasticity and network dynamics for learning in spike-based models of recurrent neural networks in the brain. So far most biologically plausible models for learning in RSNNs have focused on STDP or other synaptic plasticity mechanisms. Usually it was also assumed that this synaptic plasticity mechanisms becomes immediately effective, which is not consistent with experimental data on STDP[2]. Our results suggest that such mechanisms for synaptic plasticity are likely to be complemented with other mechanisms that especially support fast learning.

One fundamental insight that emerges from this analysis is that learning in RSNNs can be substantially more versatile and faster than previously thought. In particular, salient information during learning can also be encoded in the hidden variables of neurons if one also includes slower processes of biological neurons in the

neuron models. We have considered here only one such slow process, spike frequency adaptation of neurons, and shown that it has a remarkable impact on the learning capability of a network of spiking neurons. In particular one arrives in this way at the first spiking neural network models that can explain, through a biologically realistic neural network model, the capability of brains to learn significant behavioural improvements in very few trials, often even in a single trial. Our neural network model is based on data-based models for neurons, such as the GLIF neurons[29]. Hence we conjecture that our new learning paradigms can also be implemented and tested in such large-scale data-based models of brain areas. It particular, it opens the door for modelling concrete fast learning processes of the brain that have been analysed in previous studies[3,5]. Our model can be used to understand these and related biological phenomena, such as fast adaptation of motor predictions, see Fig. 2.

We have demonstrated two specific advantages of this new model for learning in recurrent networks of spiking neurons:

1. It substantially enlarges the diversity and power of learning strategies by which recurrent networks of spiking neurons can learn, see for example the demonstration with one shot learning of patterns by a RSNN and instantaneous deletion of an pattern in Fig. 4.
2. These networks can learn substantially faster than previously thought by making use of prior knowledge that is stored in their synaptic weights, see Figs. 1 and 3. In particular, we have shown in Fig. 1 that, once the network has learnt the overall task structure, it is able to ignore misleading information, thereby enabling robust learning from few examples[15].

We have also shown in Fig. 3 that an application of our two-tiered learning model can solve the Morris water maze task, a well known biological learning paradigm[25,26]. This task was modelled as a continuous control problem, and we applied meta-reinforcement learning to the spiking neural network. This enabled the outer loop to extract two abstract pieces of information into the spiking neural network from its interaction with the environment: that the goals are always on the perimeter of the maze, and that the goal position does not change during trials that belong to the same learning episode. The network was able to apply this abstract learnt knowledge in order to solve very fast instances of the task that it had never encountered before.

Our new model for learning in neural networks of the brain makes a clear experimentally testable predictions: It predicts that traces of fast learning become first apparent in a modified network dynamics, and only later in modifications of synaptic strengths. More specifically, our model predicts that very recently acquired information can be decoded first from the effective firing thresholds of neurons or other slowly changing variables of neurons and synapses. We expect that some of this newly acquired information is transformed and generalized during consolidation into synaptic weights[30].

Altogether our results suggest that learning in RSNNs of the brain is likely to engage other neurophysiological mechanisms besides synaptic plasticity, and that evolution, development and prior learning are likely to have configured and aligned these different processes so that they complement each other when a new learning task arises. This perspective opens the door to a much richer and functionally more powerful range of network learning methods than those which just consider synaptic plasticity. The spiking neural networks that emerged in the various tasks we considered, computed and learnt with a brain-like sparse firing activity. This is quite different from the dynamics of a spiking neural networks that operates with rate-codes. Hence these paradigms also broaden our insight into ways in which brains are able to compute and learn with sparsely active spiking neurons.

Methods

Network models

Neurons were modelled after the standard leaky integrate-and-fire (LIF) model with a proportion of neurons in all the networks consisting of LIF neurons with spike frequency adaptation (SFA) as in Bellec et al.[10], Salaj et al.[11] and described here. The use of SFA in spiking neural networks is required to get functionality comparable performance with LSTMs

Leaky integrate and fire (LIF) neurons

A LIF neuron j has one state variable – its membrane potential $V_j(t)$. Between spikes, the membrane potential $V_j(t)$ evolved according to:

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

where $I_j(t)$ is the input, and R_m is the electrical resistance term.

The neuron emitted a spike whenever the membrane potential $V_j(t)$ exceeded the threshold v_{th}. At each spike (at time t), the membrane potential $V_j(t)$ was reset by subtracting the threshold value v_{th}. After this, the neuron entered a refractory period of τ_{ref} time steps during which time it is not allowed to spike.

In discrete time, the input and output spike trains were modeled as binary sequences $x_i(t), z_j(t) \in \{0, 1\}$. Neuron j emitted a spike at time t if it was currently not in a refractory period, and its membrane potential $V_j(t)$ was above its threshold. During the refractory period following a spike, $z_j(t)$ was fixed to 0. In discrete time, using timesteps of δt, the neuron was simulated as:

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

where $\alpha = e^{-\frac{\delta t}{\tau_m}}$, τ_m is the membrane constant of the neuron j. The neuron spike is defined as $z_j(t) = H(V_j(t) - v_{\text{th}})$, where $H(x)$ is the Heaviside step function i.e. $H(x) = 1$ if $x > 0$ and 0 otherwise. In all our simulations, δt was set to 1 ms and R_m was set to 1G Ω.

The input current $I_j(t)$ in Eq. (2) was defined as the weighted sum of spikes from external inputs (x_i) and other neurons (z_i) in the network:

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

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 from neuron j to neuron i.

More complex neuron models

It is well-known that LIF neurons do not capture the internal dynamics of biological neurons very well. We used a version of generalized LIF neuron models, similar to the $GLIF_2$ neuron model of Teeter et al.[18] by using LIF neurons with spike frequency adaptation (SFA)[10,11]. To include SFA into the LIF neuron model described earlier, we replaced the fixed firing threshold v_{th} with an activity-dependent adaptive threshold $A_j(t)$. Whenever the membrane potential $V_j(t)$ exceeded this adaptive firing threshold $A_j(t)$ (instead of v_{th}), the neuron emitted a spike z_j, and its membrane potential was reset as before. But importantly, the firing threshold $A_j(t)$ was updated at every timestep in discrete time as:

$$\begin{aligned} A_j(t) &= v_{\text{th}} + \beta a_j(t), \\ a_j(t + \delta t) &= \rho_j a_j(t) + (1 - \rho_j)\, z_j(t). \end{aligned} \tag{4}$$

The new term $a(t)$ denotes the activity-dependent component of the firing threshold, $\beta > 0$ is the relative amplitude of the activity-dependent component. After each spike, $a_j(t)$ is increased by a fixed value and then decays back to 0. The parameter $\rho = e^{\frac{-\delta t}{\tau_a}}$ controls the speed by which $a(t)$ decays back to 0, where τ_a is the adaptation time constant. This overall amounts to increasing the threshold $A_j(t)$ at every spike, which then decays back to the steady state threshold v_{th} over time, the rate of decay controlled by ρ through τ_a. Adaptation time constants τ_a of neurons with SFA were chosen to match the task requirements. *Network setup* In all our experiments, we used a recurrent network of spiking neurons with a defined fraction of the neurons being LIF neurons (without SFA) and the rest being LIF neurons with SFA as shown in Fig. 5B. The specific proportion of LIF neurons with and without SFA, as well as the specific values of τ_a, and other hyper parameters such as size of the network are different for each experiment and described in Sect. "Details of the learning experiments in results". The inputs were provided to all the neurons, and all the neurons contributed to the output. The output readout was different for different experiments as described in Sect. "Details of the learning experiments in results" ("*Output decoding*" in each subsection).

When the neuron sign were not constrained (all except experiment of Fig. 3), the initial network weights were drawn from a Gaussian distribution $W_{ji} \sim \frac{w_0}{\sqrt{n_{\text{in}}}} \mathcal{N}(0, 1)$, 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[31,32] can be used to avoid tuning by hand the scaling of W_{ji}. In particular the scaling $\frac{1}{\sqrt{n_{\text{in}}}}$ used above was sufficient to initialize networks with realistic firing rates and that can be trained efficiently.

When the neuron signs were constrained (experiment of Fig. 3), all outgoing weights W_{ji}^{rec} or W_{ji}^{out} of a neuron i had the same sign. In those cases, DEEP R[27] was used as it maintains the sign of each synapse during training. The sign is thus inherited from the initialization of the network weights. To efficiently initialize weight matrices for given fractions of inhibitory and excitatory neurons, 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. A constant is added to each weight so that the sum of excitatory

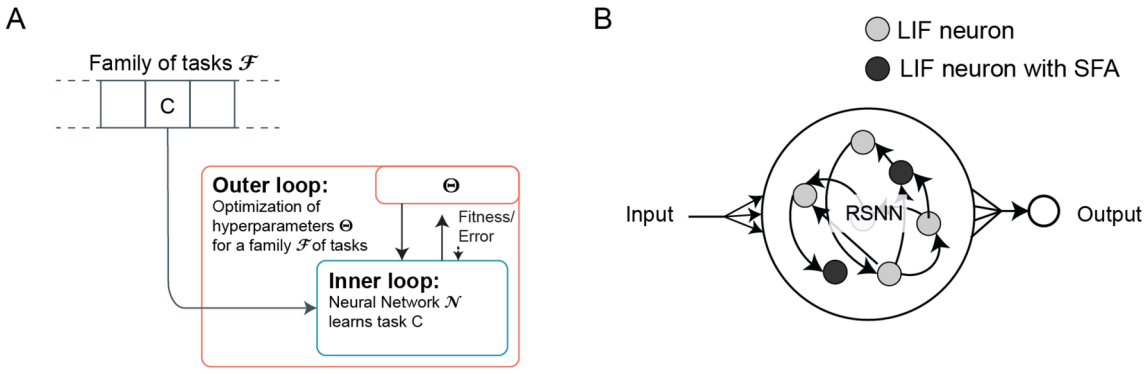

Figure 5. Schema of the learning architecture that we consider. (**A**) The two levels of optimization/learning for learning-to-learn that is used in the experiments in this paper is illustrated here. Learning by a neural network $\mathcal{N}$ is enhanced by prior optimization of hyperparameters for a large family of learning tasks. (**B**) Generic architecture of the biologically realistic neural network models $\mathcal{N}$ that we consider in all the experiments, consisting of LIF neurons both with and without SFA.

and inhibitory weights onto each neuron j ($\sum_i W_{ji}$) is zero[33] (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[31]. 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 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.

Details of the learning to learn setup

Optimization (learning) is carried out in this model at two levels as shown in Fig. 5A: The "inner loop" involves the learning of a single task by a network $\mathcal{N}$, which will be, in our case, a network of spiking neurons. The network parameters are kept fixed in the inner loop in all the experiments, and learning or adaptation happens in the dynamics of the recurrent network. The "outer loop" involves optimization of some hyperparameters Θ of the network to support fast learning of the individual tasks in the inner loop. The outer loop optimization proceeds on a much larger time scale than the inner loop, and considers a large, in general infinitely large, range of learning tasks $\mathcal{F}$ instead of a single learning task. This outer loop mimics the impact of evolutionary, developmental and prior learning processes, as well as prior learning, on parameters of the neural network $\mathcal{N}$. Notably, it does not optimize these parameters for a single learning task, but for fast learning of any generic new task C from the considered range $\mathcal{F}$ of learning tasks. This optimization is carried out in this study through backpropagation through time (BPTT) to minimize the loss on batches of different tasks chosen from the given family of learning tasks $\mathcal{F}$. Note that we use the terms training and optimization interchangeably in this paper. For simplicity, we let all synaptic weights of the RSNN $\mathcal{N}$ belong to the set of hyperparameters that are optimized in the outer loop. Hence the outer loop training shapes the activation dynamics of the RSNN, which include its firing activity and short-term memory.

Network simulation in the inner loop

In each episode of the inner loop, a task C was chosen from the family of tasks $\mathcal{F}$. The RSNN received a sequence of inputs $\mathbf{x}^k$ corresponding to this C, each encoded through the population activity of spiking neurons. In addition, it received either a cue (experiment in Sect. "New learning capabilities of recurrent networks of spiking neurons") or feedback (all other experiments) of what the target output should have been for the previously presented input $C(x^{k-1})$ (The feedback was set to zero in the first time step). The network could use the cue or delayed target feedback to adapt its behavior. The network had to predict the target $y^k = C(x^k)$ at each time step k. There was no synaptic plasticity in the inner loop.

Hyperparameter optimization in the outer loop

The outer loop optimization of learning-to-learn happened in the following way: In each iteration, a batch of different random tasks were chosen from the family $\mathcal{F}$ and the inner loop is simulated for each of these tasks by presenting the corresponding inputs to the RSNN. The predictions from the inner loop were used to compute a loss function that compared the prediction to the target for the entire batch of tasks. We used backpropagation through time (BPTT) to optimize the hyperparameters in the outer loop of L2L, which were the synaptic weights of the RSNN in our experiments.

Since the spike output of a LIF neuron model is not differentiable, we used a pseudo-derivative, but with an additional factor $\gamma < 1$ that dampens the increase of backpropagated errors through spikes as in[10,11]:

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

where $v_j(t)$ denotes the normalized membrane potential $v_j(t) = \frac{V_j(t) - A_j(t)}{A_j(t)}$. A proper choice of the dampening factor turns out to be critical in such applications of BPTT to RSNNs, since the gradient needs to propagate backwards through many layers (= time slices) of the unrolled RSNN. In neurons with SFA, gradients can be propagated efficiently through the hidden variable that denotes the dynamic threshold, without requiring a pseudo-derivative or dampening factor like for the backpropagation through spikes.

Details of the learning experiments in Results

Priors encoded in synaptic weights can significantly speed up learning

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

Input encoding. Analog values were transformed into spiking trains in exactly the same way as for the previous section.

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

RSNN setup and training schedule. The standard RSNN model was used, with 100 hidden neurons, of which 40% were LIF neurons with SFA and the rest were LIF neurons without SFA. We used all-to-all connectivity between all neurons.

The network training proceeded as follows: A new target function was randomly chosen for each episode of training, i.e. the parameters of the target function were chosen uniformly randomly from within the ranges above. Each episode consisted of a sequence of 500 steps, each lasting for 20 ms. In each step, one training example from the current function to be learned was presented to the RSNN. In such a step k, the inputs to the RSNN consisted of a randomly chosen scalar input x^k. In addition, at each step, the RSNN also got the target value $C(x^{k-1})$ from the previous step, i.e. the value of the target calculated using the target function for the inputs given at the previous step (in the first step, $C(x^0)$ is set to 0).

All the weights of the RSNN were updated using our variant of *BPTT*, once per iteration, where an iteration consists of a batch of 100 episodes, and the weight updates were accumulated across episodes in an iteration. We used the Adam optimizer[34] with the default parameters with a learning rate of 0.001. The loss function for training was the mean squared error (MSE) of the RSNN predictions over an iteration (i.e. over all the steps in an episode, and over the entire batch of episodes in an iteration):

$$\mathscr{L}(\Theta) = \mathbb{E}_{C\sim\mathscr{F}}\left[\sum_{k=1}^{K}\left(\boldsymbol{y}^k - \hat{\boldsymbol{y}}^k(\boldsymbol{x};\Theta)\right)^2 + \lambda\left(f_{\text{avg}}(\boldsymbol{x},\Theta) - f_0\right)^2\right] \tag{6}$$

where $K = 500$ is the number of steps i.e. the number of points presented to the network, each lasting for 20 ms, $\hat{\boldsymbol{y}}^k$ and $\boldsymbol{y}^k$ are the network prediction and target respectively at each step k, $\boldsymbol{x}$ is the input to the network, $\lambda = 30$ is the coefficient of the regularization term, f_{avg} is the average firing rate of the network over the entire episode, and $f_0 = 20Hz$ is the target firing rate in the regularization term. With the regularization term, we induce the RSNN to use sparse firing. We trained the RSNN for 5000 iterations.

Parameter values. The RSNN parameters were as follows: 5 ms neuronal refractory period, delays of 1 ms, adaptation time constants of the LIF neurons with SFA spread uniformly between $1 - 3000$ ms, $\beta = 1.6mV$ for LIF neurons with SFA (0 for LIF neurons without SFA), membrane time constant $\tau = 20ms$, 30 mV baseline threshold voltage. The dampening factor for training was $\gamma = 0.3$.

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

For visualizing the internal model of the RSNN, we show, for any potential input value x, the output y which the RSNN would give if this x would occur as the next network input (in a hypothetical experiment, that has no effect on the next steps of the learning process for learning the target function f). More precisely, to produce these panels, we stored the network state (i.e. the membrane potentials and all other dynamic parameters) at the corresponding time steps during the inner loop learning process. We then continued the simulation from these states with inputs from -5 to 5 and the network predictions were plotted as the orange curve in in panels Fig. 1B–E. The network state was not allowed to change when these test inputs x are shown.

Fast adaptation of motor predictions

Task family. The family of functions was defined by different two-link arms where the length and masses of the links were randomly chosen in the range [0.5, 2]. The torques were generated randomly as described in[24]. The network was trained to predict the arm state, i.e. the angles ϕ_1, ϕ_2 of its two links.

Input encoding. Analog values were transformed into spike trains to serve as inputs to the RSNN as follows: For each input component, 100 input neurons are assigned values $c_1, \ldots, c_{100}$ evenly distributed between the minimum and maximum possible value of the input. Each input neuron has a Gaussian response field with a particular mean and standard deviation, where the means are uniformly distributed between the minimum and maximum values to be encoded, and with a constant standard deviation. More precisely, the firing rate r_i (in Hz) of each input neuron i is given by $r_i = r_{\max}\ \exp\left(-\frac{(m_i - z_i)^2}{2\sigma^2}\right)$, where $r_{\max} = 200$ Hz, m_i is the value assigned to that neuron, z_i is the analog value to be encoded, and $\sigma = \frac{(m_{\max} - m_{\min})}{1000}$, $m_{\min}$ with $m_{\max}$ being the minimum and maximum values to be encoded.

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

RSNN setup and training schedule. The standard RSNN model was used, with 600 hidden neurons. Of these, 50% were LIF neurons with SFA and the rest were LIF neurons without SFA. We used all-to-all connectivity between all neurons.

The training was as follows: During inner loop training, for each episode, we randomly chose a value for the mass and length for each link of the arm. The RSNN received the motor command $c(t) = [c_1(t)\ c_2(t)]^T$, and the actual state vector of the arm $\tau = 100$ms ago, $s(t-\tau)$ as inputs. The state vector of the arm $s(t) = [\phi_1(t)\ \phi_2(t)]^T$ was defined by the angles ϕ_1, ϕ_2 of its two links. All the inputs were encoded into spikes using a population-rate code before being presented to the network (as shown in Fig. 2D top panel). A linear readout on the trace of the neural activity was used to generate the predictions of the state of the arm $\hat{s}(t;\Theta)$. Each episode lasted for 30 seconds, where the torque changed every 10 ms.

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

$$\mathscr{L}(\Theta) = \mathbb{E}_{C\sim\mathscr{F}}\left[\int_t \left(\boldsymbol{s}(t) - \widehat{\boldsymbol{s}}(t;\Theta)\right)^2 dt\right] \tag{7}$$

Parameter values. The RSNN parameters were as follows: 5 ms neuronal refractory period, delays of 1 ms, adaptation time constants of the LIF neurons with SFA spread uniformly between $1-600$ ms, $\beta = 1.7mV$ for LIF neurons with SFA (0 for LIF neurons without SFA), membrane time constant $\tau = 20ms$, 30 mV baseline threshold voltage. The dampening factor for training was $\gamma = 0.3$. We used the Adam optimizer[34] with the default parameters with a learning rate of 0.001 and a batch size of 80 for training.

Spiking neural networks can learn extremely fast from rewards — without engaging synaptic plasticity

Task family. The tasks consisted of a family of navigation tasks in a two-dimensional circular arena. For all tasks, the arena was a circle with radius 1 and goals were smaller circles of radius 0.3 with centers uniformly distributed on the circle of radius 0.85. At the beginning of an episode and after the agent reached a goal, the agent's position was set randomly with uniform probability within the arena. At every timestep, the agent chose an action by generating a small velocity vector of Euclidean norm smaller or equal to $a_{\text{scale}} = 0.02$. When the agent reached the goal, it received a reward of 1. If the agent attempted to move outside the arena, it received a negative reward of -0.02 and its new position was given by the intersection of the velocity vector with the border.

Input encoding. Information of the current environmental state $s(t)$ and the reward $r(t)$ were provided to the RSNN at each time step t as follows: The state $s(t)$ was given by the x and y coordinate of the agent's position (see top of Fig. 3C). Each position coordinate $\xi(t) \in [-1, 1]$ was encoded by 40 neurons which spiked according to a Gaussian population rate code defined as follows: a preferred coordinate value ξ_i, was assigned to each of the 40 neurons, where ξ_i's were evenly spaced between -1 and 1. The firing rate of neuron i was then given by $r_{\max}\exp(-100(\xi_i - \xi)^2)$ where $r_{\max}$ was 500 Hz. The instantaneous reward $r(t)$ was encoded by two groups of 40 neurons (see green row at the top of Fig. 3C). All neurons in the first group spiked in synchrony each time a reward of 1 was received (i.e. the goal was reached), and the second group spiked when a reward of -0.02 was received (i.e. the agent moved into a wall).

Output decoding. The output of the RSNN was provided by five readout neurons. Their membrane potentials $y_i(t)$ defined the outputs of the RSNN. The action vector $\mathbf{a}(t) = (a_x(t), a_y(t))^T$ was sampled from the distribution π_θ which depended on the network parameters θ through the readouts $y_i(t)$ as follows: The coordinate $a_x(t)$ ($a_y(t)$) was 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 updated the agent's position was then defined as $a_{scale}\,\mathbf{a}(t)$. If this velocity had a norm larger than a_{scale}, it was clipped to a norm of a_{scale}.

The last readout output $y_5(t)$ was used to predict the value function $V_\theta(t)$. It estimated 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 $\frac{1}{R_m}v_j$ to the current I_j into neuron j. Here, v_j is a network parameter initialized at 0.03 and optimized by *BPTT* alongside the network weights.

RSNN setup and training schedule. To train the network we used the Proximal Policy Optimization algorithm (PPO)[35]. For each training iteration, K full episodes of T timesteps were generated with fixed parameters θ_{old} (here $K = 10$ and $T = 2000$). We write the clipped surrogate objective of PPO as $O^{PPO}(\theta_{old}, \theta, t, k)$ (this is defined under the notation L^{CLIP} in[35]). The loss with respect to θ was then defined as follows:

$$\mathscr{L}(\theta) = -\frac{1}{KT}\sum_{k<K}\sum_{t<T} O^{PPO}(\theta_{old}, \theta, t, k) + \mu_v(R(t,k) - V_\theta(t,k))^2 \tag{8}$$

$$- \mu_e H(\pi_\theta(k,t)) + \mu_{firing}\left(f_{\text{avg}}(\boldsymbol{x},\Theta) - f_0\right)^2, \tag{9}$$

where $H(\pi_\theta)$ is the entropy of the distribution π_θ, f^0 is a target firing rate of 10 Hz, $\mu_v, \mu_e, \mu_{firing}$ are regularization hyper-parameters, f_{avg} is the average firing rate of the network over the entire episode. Importantly probability distributions used in the definition of the loss $\mathscr{L}$ (i.e. the trajectories) were conditioned on the current noises, so that for the same noise and infinitely small parameter change from θ_{old} to θ the trajectories and the spike trains were the same. At each iteration this loss function $\mathscr{L}$ was then minimized with one step of the ADAM optimizer[34].

Parameter values. In this task the RSNN had 400 hidden units (200 excitatory LIF neurons, 80 inhibitory LIF neurons and 120 LIF neurons with SFA with adaptation time constants $\tau_a = 1200$ ms) and the network was rewired with a fixed global connectivity of 20% using DEEP R[27]. DEEP R provides a way to train a sparsely connected neural network directly using BPPT, while maintaining a fixed overall sparsity in the network and fixed sign for each of the connections. The latter ability allows us to train a network that obey Dale's law with fixed excitatory and inhibitory neurons. 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}.

New learning capabilities of recurrent networks of spiking neurons

Task family. In each task C from the family of tasks $\mathscr{F}$, three 25-bit randomly generated patterns were shown in the first phase referred to as phase A (see Fig. 4 for illustration of phases). Each of these three patterns were presented to the network for 100 ms. In the next phase B, partial versions of the same three patterns were shown. The partial patterns were generated by setting each non-zero bit in each pattern to zero with 40% probability. The targets for this phase were the full patterns from phase A. In phase C, a cue signal was given to denote to the network which of the three patterns should be "deleted" by index. In the last phase D, the same partial patterns as in phase B were shown again. The targets were the full patterns for the two patterns that were not "deleted". For the pattern that was "deleted", the target was one of other two patterns that was closest to the deleted pattern measured by hamming distance.

Input encoding. The patterns were generated as 25-bit vectors, and the cue as a 3-bit vector. In both cases, each bit was represented by 5 spiking neurons, which fired at a high rate of 200 Hz when the bit was 1, and at a lower rate of 2Hz when it was 0.

Output decoding. The output of the RSNN was a linear readout that received as input the trace of the firing of all the neurons in the network. The spiking activity of the neurons was convolved with an exponential kernel with time constant 100 ms to generate the trace.

RSNN setup and training schedule. The standard RSNN architecture was used, with 300 hidden neurons. Of these, 50% were LIF neurons with SFA and the rest were LIF neurons without SFA. We used all-to-all connectivity between all neurons.

The network training proceeded as follows: In each episode, a new set of three random patterns were chosen along with the pattern that was to be excluded from the output. These were presented to the network as described above.

All the weights of the RSNN were updated using our variant of *BPTT*, once per iteration, where an iteration consists of a batch of 100 episodes, and the weight updates were accumulated across episodes in an iteration. We used the Adam optimizer[34] with the default parameters with a learning rate of 0.001. The loss function for training was the bit-wise cross-entropy loss of the RSNN predictions:

$$\mathscr{C}\ \mathscr{E}\ (\boldsymbol{y}^k, \boldsymbol{p}^k) = -\frac{1}{N}\sum_{n=1}^{N}\left(y^{k,n}\ \log(p^{k,n}) - (1-y^{k,n})\log(1-p^{k,n})\right) \tag{10}$$

where $\mathscr{C}\ \mathscr{E}$ is the cross-entropy loss, $\boldsymbol{y}^k$ and $\boldsymbol{p}^k$ are, respectively, the target 25-bit vector and vector of predicted probability of each of the $N = 25$ bits being 1 at step k.

The overall loss was given by:

$$\mathscr{L}(\Theta) = \mathbb{E}_{C\sim\mathscr{F}}\left[\frac{1}{K}\sum_{k=1}^{K}\mathscr{C}\ \mathscr{E}\ (\boldsymbol{y}_B^k, \boldsymbol{p}_B^k(\boldsymbol{x},\Theta)) + \frac{1}{K}\sum_{k=1}^{K}\mathscr{C}\ \mathscr{E}\ (\boldsymbol{y}_D^k, \boldsymbol{p}_D^k(\boldsymbol{x},\Theta)) + \lambda\left(f_{\text{avg}}(\boldsymbol{x},\Theta) - f_0\right)^2\right] \tag{11}$$

where $K = 3$ is the number of patterns shown in each phase, and the subscripts B and D denote phase to which the target and prediction vectors correspond, $\lambda = 5$ is the coefficient of the regularization term, f_{avg} is the average firing rate of the network over the entire episode, and $f_0 = 20Hz$ is the target firing rate in the regularization term. With the regularization term, we induce the RSNN to use sparse firing. We trained the RSNN for 100,000 iterations.

Parameter values. The RSNN parameters were as follows: 5 ms neuronal refractory period, delays of 1 ms, adaptation time constants of the LIF neurons with SFA were spread uniformly between $1 - 1000$ ms, $\beta = 1.7$ mV for LIF neurons with SFA (0 for LIF neurons without SFA), membrane time constant $\tau = 20ms$, 30 mV baseline threshold voltage. The dampening factor for training was $\gamma = 0.3$.

Data availability

The datasets used and/or analysed during the current study are available from the corresponding author on reasonable request. All datasets are generated programmatically, and the details of how to generate the data are included in the manuscript.

Code availability

All the code for the experiments is available in the Supplementary Information, and will be made public on publication at the following url: https://github.com/anandtrex/fast-snn-learning.

Received: 8 June 2023; Accepted: 27 February 2024
Published online: 12 April 2024

References

1. Brea, J. & Gerstner, W. Does computational neuroscience need new synaptic learning paradigms?. *Curr. Opin. Behav. Sci.* **11**, 61–66. https://doi.org/10.1016/j.cobeha.2016.05.012 (2016).
2. Froemke, R. C., Debanne, D. & Bi, G.-Q. Temporal modulation of spike-timing-dependent plasticity. *Front. Synaptic Neurosci.* **2**, 19 (2010).
3. Perich, M. G., Gallego, J. A. & Miller, L. E. A neural population mechanism for rapid learning. *Neuron* **100**, 964–976 (2018).
4. Botvinick, M. *et al.* Reinforcement learning, fast and slow. *Trends Cogn. Sci.* **23**, 408–422. https://doi.org/10.1016/j.tics.2019.02.006 (2019).
5. Crevecoeur, F., Mathew, J., Bastin, M. & Lefevre, P. Feedback adaptation to unpredictable force fields in 250 ms. *eNeuro*https://doi.org/10.1523/ENEURO.0400-19.2020 *(2020).*
6. Hochreiter, S., Younger, A. S. & Conwell, P. R. Learning to learn using gradient descent. In *International Conference on Artificial Neural Networks* (eds Hochreiter, S. *et al.*) 87–94 (Springer, 2001).
7. Duan, Y. *et al.*RL^2: Fast reinforcement learning via slow reinforcement learning. Preprint at http://arxiv.org/abs/1611.02779 (2016).
8. Wang, J. X. *et al.* Learning to reinforcement learn. Preprint at http://arxiv.org/abs/1611.05763 (2016).
9. Wang, J. X. *et al.* Prefrontal cortex as a meta-reinforcement learning system. *Nat. Neurosci.* **21**, 860–868 (2018).
10. Bellec, G., Salaj, D., Subramoney, A., Legenstein, R. & Maass, W. Long short-term memory and Learning-to-learn in networks of spiking neurons. *Adv. Neural Inf. Process. Syst.* **31**, 795–805 (2018).
11. Salaj, D. *et al.* Spike frequency adaptation supports network computations on temporally dispersed information. *eLife* **10**, e65459. https://doi.org/10.7554/eLife.65459 (2021).
12. Marder, E., Abbott, L., Turrigiano, G. G., Liu, Z. & Golowasch, J. Memory from the dynamics of intrinsic membrane currents. *Proc. Natl. Acad. Sci.* **93**, 13481–13486 (1996).
13. Turrigiano, G. G., Marder, E. & Abbott, L. Cellular short-term memory from a slow potassium conductance. *J. Neurophysiol.* **75**, 963–966 (1996).
14. Gutkin, B. & Zeldenrust, F. Spike frequency adaptation. *Scholarpedia* **9**, 30643. https://doi.org/10.4249/scholarpedia.30643 (2014).
15. Gershman, S. J. & Niv, Y. Learning latent structure: Carving nature at its joints. *Curr. Opin. Neurobiol.* **20**, 251–256. https://doi.org/10.1016/j.conb.2010.02.008 (2010).
16. Harlow, H. F. The formation of learning sets. *Psychol. Rev.* **56**, 51 (1949).
17. Zador, A. M. A critique of pure learning and what artificial neural networks can learn from animal brains. *Nat. Commun.* **10**, 1–7. https://doi.org/10.1038/s41467-019-11786-6 (2019).
18. Teeter, C. *et al.* Generalized leaky integrate-and-fire models classify multiple neuron types. *Nat. Commun.* **1**, 1–15 (2018).
19. Allen Institute for Brain Science. Allen Cell Types Database, cell feature search. celltypes.brain-map.org/data (2018).
20. Crevecoeur, F., Thonnard, J.-L. & Lefevre, P. A very fast time scale of human motor adaptation: Within movement adjustments of internal representations during reaching. *eNeuro*https://doi.org/10.1523/ENEURO.0149-19.2019 *(2020).*
21. Lalazar, H. & Vaadia, E. Neural basis of sensorimotor learning: Modifying internal models. *Curr. Opin. Neurobiol.* **18**, 573–581 (2008).
22. Wolpert, D. M. & Ghahramani, Z. Computational principles of movement neuroscience. *Nat. Neurosci.* **3**, 1212 (2000).
23. Wong, J. D., Kistemaker, D. A., Chin, A. & Gribble, P. L. Can proprioceptive training improve motor learning?. *J. Neurophysiol.* **108**, 3313–3321 (2012).
24. Gilra, A. & Gerstner, W. Predicting non-linear dynamics by stable local learning in a recurrent spiking neural network. *Elife* **6**, e28295 (2017).
25. Morris, R. Developments of a water-maze procedure for studying spatial learning in the rat. *J. Neurosci. Methods* **11**, 47–60 (1984).
26. Vasilaki, E., Frémaux, N., Urbanczik, R., Senn, W. & Gerstner, W. Spike-based reinforcement learning in continuous state and action space: when policy gradient methods fail. *PLoS Comput. Biol.* **5**, e1000586 (2009).
27. Bellec, G., Kappel, D., Maass, W. & Legenstein, R. Deep rewiring: Training very sparse deep networks. In *International Conference on Learning Representations (ICLR)* (2018).
28. Hopfield, J. J. Neural networks and physical systems with emergent collective computational abilities. *Proc. Natl. Acad. Sci.* **79**, 2554–2558. https://doi.org/10.1073/pnas.79.8.2554 (1982).
29. Billeh, Y. N. *et al.* Systematic integration of structural and functional data into multi-scale models of mouse primary visual cortex. *Neuron* **106**, 388-403.e18. https://doi.org/10.1016/j.neuron.2020.01.040 (2020).
30. Stickgold, R. & Walker, M. P. Sleep-dependent memory triage: Evolving generalization through selective processing. *Nat. Neurosci.* **16**, 139–145. https://doi.org/10.1038/nn.3303 (2013).
31. Bengio, Y., Simard, P. & Frasconi, P. Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. Neural Netw.* **5**, 157–166 (1994).
32. Sussillo, D. & Abbott, L. Random walk initialization for training very deep feedforward networks. Preprint at http://arxiv.org/abs/1412.6558 (2014).
33. Rajan, K. & Abbott, L. F. Eigenvalue spectra of random matrices for neural networks. *Phys. Rev. Lett.* **97**, 188104 (2006).
34. Kingma, D. P. & Ba, J. Adam: A method for stochastic optimization. Preprint at http://arxiv.org/abs/1412.6980 (2014).
35. Schulman, J., Wolski, F., Dhariwal, P., Radford, A. & Klimov, O. Proximal policy optimization algorithms. Preprint at http://arxiv.org/abs/1707.06347 (2017).

Acknowledgements

This research was partially supported by the Human Brain Project (Grant Agreement number 785907) of the European Union and by the ERA-NET CHIST-ERA programme by the Austrian Science Fund (FWF) (project SMALL, project number I 4670-N). We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Quadro P6000 GPU used for this research. Computations were carried out on the Human Brain Project PCP Pilot Systems at the Juelich Supercomputing Centre, which received co-funding from the European Union (Grant Agreement number 604102) and on the Vienna Scientific Cluster (VSC).

Author contributions

A.S., G.B., and W.M. conceived the work, A.S., G.B. and F.S. carried out experiments, A.S, G.B., R.L. and W.M. contributed to the writing of the paper. All authors reviewed the manuscript.

Competing interests

The authors declare no competing interests.

Additional information

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1038/s41598-024-55769-0.

Correspondence and requests for materials should be addressed to W.M.

Reprints and permissions information is available at www.nature.com/reprints.

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.