

RESEARCH ARTICLE

Non-synaptic plasticity enables memory-dependent local learning

Romain Ferrand^{1*}, Maximilian Baronig¹, Florian Unger¹, Robert Legenstein¹

Institute of Theoretical Computer Science, Graz University of Technology, Graz, Austria

* romain.ferrand@igi.tugraz.at



Abstract

Synaptic plasticity is essential for memory formation and learning in the brain. In addition, recent results indicate that non-synaptic plasticity processes such as the regulation of neuronal membrane properties contribute to memory formation, its functional role in memory and learning has however remained elusive. Here, we propose that non-synaptic and synaptic plasticity are both essential components to enable memory-dependent processing in neuronal networks. While the former acts on a fast timescale for rapid information storage, the latter shapes network processing on a slower timescale to harness this memory as a functional component. We analyse this concept in a network model where pyramidal neurons regulate their apical trunk excitability in a Hebbian manner. We find that local synaptic plasticity rules can be derived for this model and show that the interplay between this synaptic plasticity and the non-synaptic trunk plasticity enables the model to successfully accommodate memory-dependent processing capabilities in a number of tasks, ranging from simple memory tests to question answering.

OPEN ACCESS

Citation: Ferrand R, Baronig M, Unger F, Legenstein R. (2025) Non-synaptic plasticity enables memory-dependent local learning. PLoS ONE 20(3): e0313331. <https://doi.org/10.1371/journal.pone.0313331>

Editor: Gennady S. Cymbalyuk, Georgia State University, UNITED STATES OF AMERICA

Received: June 25, 2024

Accepted: October 22, 2024

Published: March 17, 2025

Peer Review History: PLOS recognizes the benefits of transparency in the peer review process; therefore, we enable the publication of all of the content of peer review and author responses alongside final, published articles. The editorial history of this article is available here: <https://doi.org/10.1371/journal.pone.0313331>

Copyright: © 2025 Ferrand et al. This is an open access article distributed under the terms of the [Creative Commons Attribution License](https://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

Data availability statement: All relevant data are within the manuscript and its [Supporting Information](#) files.

Introduction

Memory is an integral component of biological neuronal systems. It underlies behavior at many levels, starting from basic fear memory to complex cognitive processes such as language understanding. Experimental results have provided ample evidence that memories are stored in so-called memory engrams. The main assumption of the memory engram theory is that learning induces persistent changes in specific brain cells that retain information and are subsequently reactivated upon appropriate retrieval conditions [1–3].

A host of experimental evidence supports the hypothesis that synaptic plasticity is essential for memory storage. However, some recent results indicate that also non-synaptic plasticity such as the regulation of neuronal membrane properties contributes to the creation of memory engrams [4–8]. In fact, there has been some scepticism about the role of synaptic plasticity in memory formation [6,9,10]. One important argument is that synaptic plasticity, in particular long-term potentiation (LTP) typically requires repetitive stimulation, whereas learning and memory can arise from single experiences. Another suggestion was that non-synaptic plasticity may act as a permissive signal for synaptic changes [4]. In this view, rapid excitability changes of neurons could set the stage for later synaptic reconfiguration. An intriguing

Funding: This work was supported by the European Community's Horizon 2020 FET-Open Programme, grant number 899265, ADOPD. The Funder did not play any role in the study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Competing interests: The authors have declared that no competing interests exist.

experimental finding was that excitability changes are crucial for learning in trace conditioning experiments [11,12], which was shown to synergize with synaptic plasticity in a modelling study [13]. Taken together, the available studies support the intriguing hypothesis that synaptic and non-synaptic plasticity co-operate to construct memory engrams. However, as stated in [4], the question about the functional role of the plasticity of neuronal membrane characteristics has remained open. In this article, we show that fast changes of the intrinsic excitability in the apical dendritic trunk, which we call trunk strength plasticity (TSP) in the following, can give rise to enhanced learning capabilities of neuronal networks. Our hypothesis is that this non-synaptic TSP, which can rapidly be induced, provides the network with instantaneous memory. This memory is used by the network for memory-dependent computation. Synaptic plasticity on the other hand operates on a slower timescale. The role of synaptic plasticity is thus to learn to make use of the memorization capabilities introduced by TSP for the computational task at hand. In that sense, the two plasticity processes synergize in the search for a task solution (see e.g., [14–16] for other synergistic plasticity approaches).

Our theoretical analysis shows that this division of labor gives rise to temporally and spatially local synaptic plasticity rules. We find that despite the locality of the synaptic and non-synaptic plasticity processes, networks equipped with such mechanisms exhibit remarkable learning capabilities. This is demonstrated for reward-based learning tasks that necessitate instantaneous memory such as a radial maze task as well as for a complex question-answering task [17], thus illustrating that basic language understanding can be acquired through local learning.

Results

A model for memory processing in pyramidal neurons

Pyramidal neurons are the principal cells of many memory related brain areas such as the hippocampus, amygdala, and prefrontal cortex. They have a characteristic bipolar shape, with basal dendrites close to the soma and apical dendrites that are separated from the soma by an extended apical trunk (green schematic cell in Fig 1A). These two components establish two sites of synaptic integration. They independently integrate synaptic input in a non-linear manner, and these two signals are then combined at the soma to determine the action potential output of the neuron [18,19].

For our network model, we were inspired by memory-augmented neural networks (MANNs) as e.g. in [20,21]. These models use a memory module where key vectors from $\mathbb{R}^m$ can be associated with value vectors from $\mathbb{R}^m$. When some fact is stored in memory (memorization operation), a key and a value vector is computed and this key-value pair is memorized. When some information is retrieved from memory (memory recall operation), a query vector is computed and a value vector is retrieved based on the similarity between the query vector and all stored key vectors.

The retrieved value vector is then used in a single layer neural network to compute the network output. In our model, memories are stored in a *memory layer* that is a population of m model pyramidal neurons, see Fig 1B.

The memory layer receives input from two input populations with activities $\mathbf{x}^{a,t} \in \mathbb{R}^d$ and $\mathbf{x}^{b,t} \in \mathbb{R}^d$ at time t , which project to the apical and basal compartments of the pyramidal neurons via synaptic weight matrices W^{apical} and W^{basal} respectively. In each pyramidal neuron i of the memory layer, the resulting basal activation b_i^t and apical activation a_i^t is given by

$$b_i^t = \sigma(\bar{b}_i^t), \quad a_i^t = \sigma(\bar{a}_i^t), \quad (1)$$

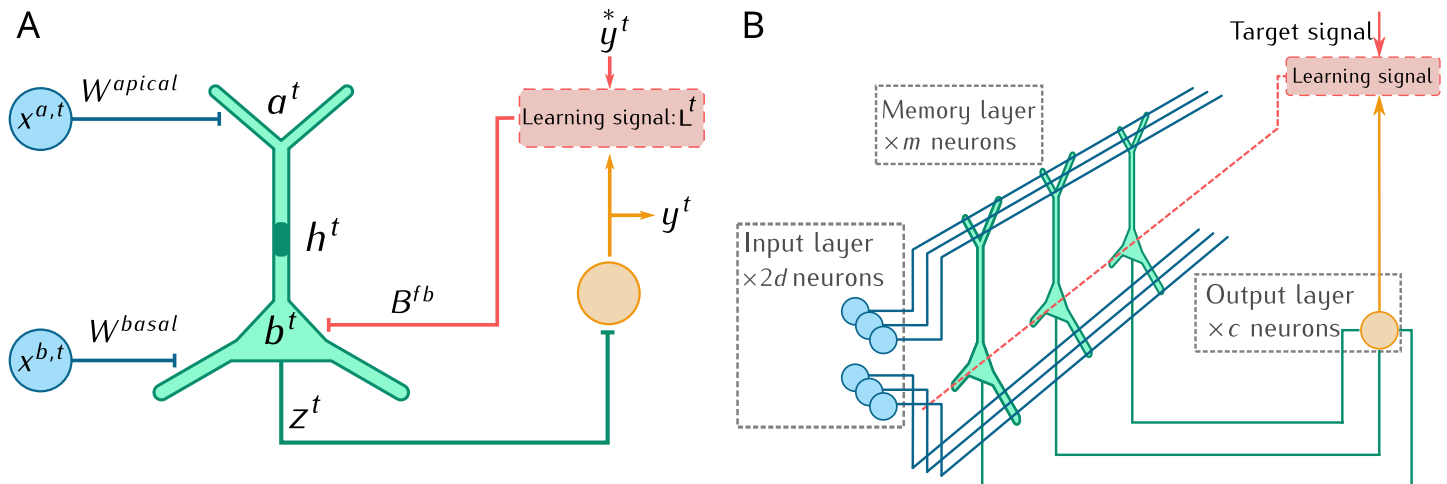


Fig 1. Network model. (A) Schematic of our simple pyramidal cell model (green) consisting of an apical and basal compartment with activations a^t and b^t respectively. The excitability of the apical trunk h^t is variable and indicated in dark green. The neuron output z^t is projected to an output layer (yellow). Prediction errors generate learning signals which are fed back via randomly initialized feedback weights B^{fb} . (B) Network view. Input neurons (blue) project to apical and basal compartments of a population of pyramidal cells (green). Pyramidal output z^t is projected to a layer of linear output neurons (yellow) producing the network output y^t .

<https://doi.org/10.1371/journal.pone.0313331.g001>

with the rectified linear nonlinearity, $\sigma(s) = \max\{0, s\}$, and pre-activations

$$\bar{b}_i^t = \sum_j w_{ij}^{\text{basal}} x_j^{b,t}, \quad \bar{a}_i^t = \sum_j w_{ij}^{\text{apical}} x_j^{a,t}. \quad (2)$$

The vector $\mathbf{a}^t \in \mathbb{R}^m$ of apical activations of neurons is thus akin to the key vector and the vector $\mathbf{b}^t \in \mathbb{R}^m$ of the basal activations is akin to the value vector in a MANN. In the following, we will suppress the neuron index i in our model description in order to simplify notation. The real valued output z^t of a pyramidal neuron is then given by a linear combination of these activations,

$$z^t = b^t + h^t a^t, \quad (3)$$

where the scalar $h^t > 0$ denotes the branch strength of the apical trunk. As we will describe below in detail, we model the trunk strength as a dynamic variable h^t , used to memorize information about the apical and basal activations (i.e., the key and value vectors). Hence, this layer of pyramidal cells implements a simple memory module. The output $\mathbf{z}^t \in \mathbb{R}^m$ of the pyramidal cell layer is projected to c output neurons (the output layer) via the weight matrix $W^{\text{out}} \in \mathbb{R}^{c \times m}$, which produces the network output $\mathbf{y}^t \in \mathbb{R}^c$. In addition, the network output can be used to determine learning signals L^t that are used as feedback signals for learning (red in Fig 1; see below).

A large variety of non-synaptic plasticity mechanisms exist in pyramidal neurons [22–26]. Changes of neuronal membrane characteristics are not necessarily only changing the global neuronal excitability or the action potential initiation threshold, they can also be confined to dendritic subunits [23,26]. In particular, Losonczy et al. found that local branch activity led to the potentiation of the branch strength when paired either with the cholinergic agonist carbachol or — more importantly — with two or three backpropagating action potentials, indicating an intra-neuron Hebbian type of branch plasticity [26]. In addition, experimental work [23] has shown that I_h currents can be down-regulated in a Hebbian manner in CA1 pyramidal neurons. Such down-regulation can in turn increase the dendro-somatic coupling

of the apical dendrites [27]. Motivated by this finding, we assume a Hebbian-type trunk plasticity:

$$\Delta h^t = \gamma_+ (h^{\max} - h^{t-1}) b^t a^t - \gamma_- h^{t-1} (a^t)^2, \quad (4)$$

$$\bar{h}^t = h^{t-1} + \Delta h^t, \quad (5)$$

$$h^t = \sigma(\bar{h}^t), \quad (6)$$

with σ the ReLU non-linearity, parameters $\gamma_+, \gamma_- > 0$, and $h^{\max} > 0$ is the maximum trunk strength.

Hence, in our model, the trunk strength is potentiated when both the apical and basal compartments are activated by synaptic input (term $b^t a^t$ in Eq (4)). This potentiation saturates at $h^{\max}$. The ReLU nonlinearity in Eq (6) assures that the trunk strength remains non-negative. We have also included a depression term that depends quadratically on the apical activation (last term in Eq (4)) similar to Oja's Hebbian rule [28]. The depression term can be interpreted as a homeostatic term as it reduces the trunk excitability when there is large activity [25]. In the particular implementation that we considered, the depression depends quadratically on the apical activity. This homeostasis can reset a memory by depressing trunk strengths of neurons where an apical activation is not paired with a basal one. Other options would be to use the basal activity or a combination of both for homeostasis. It turns out that the actual choice is not crucial as long as there is homeostasis that depends on some activation, see Section A in S1 Appendix. From a functional perspective, this plasticity enables the neurons in the memory layer to implement a simple memory system.

At each time step t , the network can perform either a memorization or a memory recall operation. In a memorization operation, both input populations $\mathbf{x}^{a,t}$ and $\mathbf{x}^{b,t}$ are active, and the trunk strength is updated. The layer thus memorizes at which neurons both the apical and basal dendrites were activated. At a memory recall, we assume that only the apical input population $\mathbf{x}^{a,t}$ is active. The activated apical compartments will induce neuronal activity only in those cells in which the trunk strength was potentiated, thus reading out a trace of the memory. The output of the memory layer activates the neurons in the output layer, a learning signal is computed, fed back to the memory layer and synaptic weights are updated according to the plasticity rules described below. Note that the network architecture is purely feed-forward, without any recurrent synaptic connections. The trunk strength however implements an implicit recurrence, see Eq (4). This dynamic state variable can be used by the network to store information about previous inputs.

In principle, the apical and basal input populations could provide different aspects of the input at a memorization event. For simplicity however, in our simulations they exhibited the same activity pattern $\mathbf{x}^{a,t} = \mathbf{x}^{b,t}$ at memorization events, which turned out to work well in our simulations.

Local synaptic plasticity for memory-dependent processing with TSP

In our model, memory is encoded in the network in the vector of trunk strengths $\mathbf{h}^t$. This memory is constructed on the fast timescale of single neuronal activations. According to our hypothesis, synaptic plasticity in contrast works on a slower timescale and is used to learn how to make use of this memory process in the specific task context. Similar ideas have been put forward in a number of models termed memory-augmented neural networks [20,21, 29,30]. In these networks however, synaptic weights are optimized with backpropagation through time (BPTT), a complex and biologically highly implausible learning algorithm [31]. In contrast, it turns out that if trunk strengths are used for rapid information storage, one can

derive local synaptic plasticity rules, see *Methods*. The derivation takes advantage of the eligibility propagation (e-prop) algorithm [32]. The resulting on-line plasticity rules approximate BPTT using synaptic eligibility traces [33] in combination with learning signals that are directly fed back from the network output to the network neurons. This is illustrated in Fig 1 for a supervised learning scenario (red). Consider a network with c output neurons and corresponding outputs $y_1^t, \dots, y_c^t$. For given target outputs $y_1^{*,t}, \dots, y_c^{*,t}$, we obtain c learning signals $l_1^t, \dots, l_c^t$ which are fed back to the memory layer neurons through a feedback weight matrix $\mathbf{B}^{\text{fb}} \in \mathbb{R}^{m \times c}$.

To simplify notation, we express the following equations for a single neuron and drop the post-synaptic neuron index. Consider a neuron in the memory layer with feedback weight vector $\mathbf{b}^{\text{fb}}$. The weighted learning signals are summed to obtain the neuron-specific learning signal L^t :

$$L^t = \sum_{k=1}^c b_k^{\text{fb}} l_k^t. \quad (7)$$

The synaptic plasticity rules for synapses at the basal and apical dendrites are then given by (see *Methods* for a derivation)

$$\Delta w_j^{\text{basal}} = \eta L^t a^t e_j^{\text{basal},t}, \quad (8)$$

$$\Delta w_j^{\text{apical}} = \eta L^t \left(a^t e_j^{\text{apical},t} + h^t H(\bar{a}^t) x_j^{a,t} \right), \quad (9)$$

where η is a learning rate, $e_j^{\text{basal},t}$ and $e_j^{\text{apical},t}$ denote the eligibility traces of the basal and apical synapses respectively, and H is the Heaviside step function: $H(s) = 0$ for $s \leq 0$ and $H(s) = 1$ otherwise.

In general, the plasticity rules combine the neuron-specific learning signal L^t , that is the assigned error to the neuron, with the synapse specific eligibility trace. This eligibility trace records the eligibility of the synapse for changes in the trunk strength. For example, if there was a large error assigned to the neuron and its trunk strength was high, synapses that led to a trunk strength increase are eligible for that error and are changed such that this increase will be smaller in future. The eligibility is computed by filtering information locally available to the neuron:

$$e_j^{\theta,t} = e_j^{\theta,t-1} f(h^t, b^t, a^t) + g^{\theta}(h^t, h^{t-1}, b^t, a^t) x_j^{\theta,t}, \quad (10)$$

for $\theta \in \{\text{basal}, \text{apical}\}$ and functions f, g^{θ} . Note that the update rules (8) and (9) as well as the updates of eligibility traces (10) need only temporally and spatially local information, that is, information that is available at the post-synaptic neuron at the current or previous time step. Hence, this update could in principle be implemented by pyramidal neurons.

The function f in Eq (10) dynamically controls the decay of the eligibility, and it is the same for both the apical and the basal synapses:

$$f(h^t, b^t, a^t) = H(\bar{h}^t) (1 - \gamma_+ b^t a^t - \gamma_- (a^t)^2), \quad (11)$$

where H denotes the Heaviside step function. Hence, f (as well as the other functions, see below) are gated by h^t , meaning that the eligibility trace is zero if the trunk strength is zero. This arises from the fact that the apical dendrite has no influence on the neuron output when $h^t = 0$, and thus the network output does not depend on the input to this neuron during a recall (note that during a recall, only the apical inputs are active). It is 1 (no decay) in the

absence of apical activity and reduced by the squared apical activity and the product of basal and apical activity. In other words, the synapse stays fully eligible as long as the trunk strength is not altered. If the trunk strength is altered due to apical activity, the eligibility is reduced, as other synapses may become eligible for these changes.

The g^θ functions in Eq (10) modulate the increase of the eligibility trace at synaptic input:

$$g^{\text{basal}}(h^t, h^{t-1}, b^t, a^t) = H(\tilde{h}^t)H(\tilde{b}^t)\gamma_+(h^{\text{max}} - h^{t-1})a^t, \quad (12)$$

$$g^{\text{apical}}(h^t, h^{t-1}, b^t, a^t) = H(\tilde{h}^t)H(\tilde{a}^t)\gamma_+[(h^{\text{max}} - h^{t-1})b^t - 2\gamma_-h^{t-1}a^t]. \quad (13)$$

Both g^{basal} and g^{apical} are gated by activity in their corresponding compartments. Hence, if the compartment is inactive, the eligibility is not changed even if there is synaptic input, as the compartment did not contribute to any changes of the trunk strength. Also, both include a term $h^{\text{max}} - h^{t-1}$ which takes into account that it is harder to increase the trunk strength when it is already close to its maximum value. g^{basal} is then linearly dependent on the apical activation, as a potentiation of the trunk strength is proportional to apical activity (Eq (4)). g^{apical} is similar, with an additional term that records eligibility according to the decrease of the trunk strength due to apical activation.

The update dynamics of eligibility traces and weights of basal synapses are illustrated in Fig 2 (in the setup shown there, the apical dynamics are similar). Here, a neuron receives two apical inputs $x_1^{a,t}, x_2^{a,t}$ and two basal inputs $x_1^{b,t}, x_2^{b,t}$ (Fig 2 right and panel A). First, $x_1^{a,t}$ and $x_1^{b,t}$ are co-active leading to an increase in the trunk strength (panel B). The eligibility-trace for the first basal synapse is thus increased as it caused this trunk potentiation (panel C). Later, there is a longer co-activation of the other synapse pair $x_2^{a,t}$ and $x_2^{b,t}$, leading to stronger trunk potentiation. Now, the second basal synapse increases its eligibility trace, while the trace of the first one is decreased. At the final time step, a learning signal appears, paired with apical activity. This leads to weight changes that are proportional to the eligibility trace (panel E).

In summary, the eligibility traces record first and second order terms of apical and basal activity, together with some gating and a dependence on the trunk strength. Although the update rules for the traces are far from simple, they include only terms available at the post-synaptic neuron and in particular are local in time — in contrast to BPTT. Hence, they can in principle be computed at the synapse.

When we compare the synaptic and non-synaptic plasticity mechanism in our model, we observe that they operate on very different timescales with complementary functions. TSP acts very fast in order to memorize relevant information about the current input in the trunk strengths of the memory layer neurons. On the other hand, the local synaptic plasticity dynamics approximate gradient descent and optimize the synaptic weights over many learning episodes in order to minimize network error on the specific task. Their roles are thus complementary, but they synergize in the following way. TSP provides general memory capabilities that help to learn rather arbitrary memory-related tasks. Synaptic plasticity then utilizes this memory and adapts synaptic weights such that the relevant information is memorized and retrieved for the task at hand. In the simulations reported below, we turned off synaptic plasticity in the testing phase after the model was trained. This shows that while the synaptic plasticity is needed for task acquisition, it is not necessary for inference after proper synaptic weights have been determined.

In the following, we evaluated the above described network on a variety of sequence processing tasks that all require some form of memory. In each task, one episode consists of a sequence of time steps t , in each of which the trunk strength h^t and the eligibility traces $e_j^{\theta,t}$ get updated in each neuron of the memory layer. Only after the entire episode, synaptic

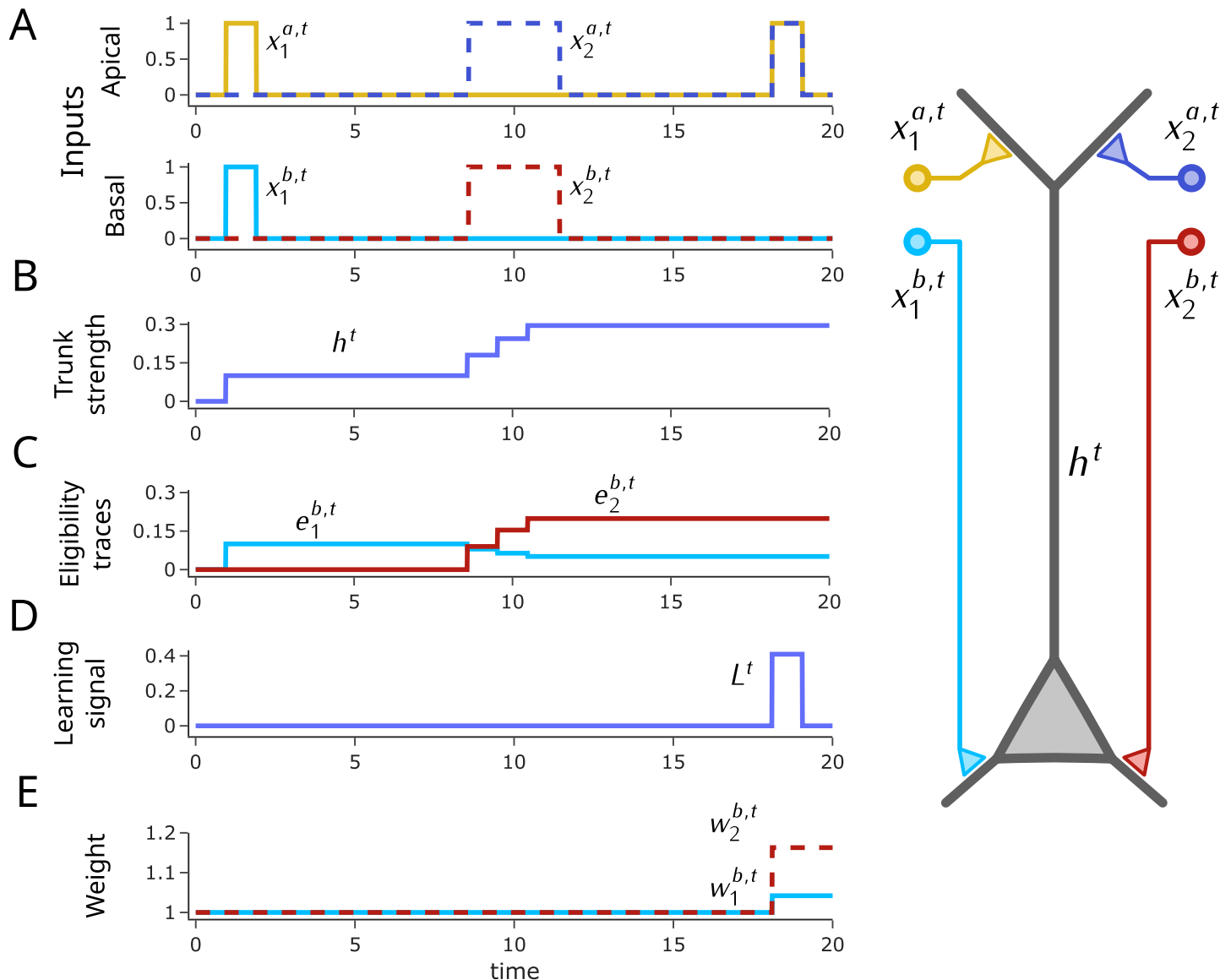


Fig 2. Illustration of eligibility trace dynamics and synaptic plasticity at the basal compartment in our model. Simulation of a single neuron with two apical and basal synapses, each having unit weight. The first apical and basal synapse is initially activated for one time step, followed by the activation of the second apical and basal synapse for three consecutive time steps (panel A). These co-activations lead to increases in the branch strength h^t (panel B), as well as to changes in the eligibility traces (panel C, see text). Then, a recall is performed at time step 19 where both apical synapses are activated. Further, a learning signal L^t is received (panel D). Changes of basal weights are then given by the product of the eligibility traces with the learning signal and apical activation (panel E, see Eq (8)).

<https://doi.org/10.1371/journal.pone.0313331.g002>

weight changes are applied based on the accumulated eligibility traces, if the model is in the training phase. For the testing phase, eligibility traces are not recorded, since no synaptic weight update is performed. The eligibility traces can hence be viewed as auxiliary functions, keeping track of the network activity over time for an accurate approximation of BPTT. Note, that eligibility traces have no direct influence on the network output, hence, changes in trunk strength are the only available resource to account for within-episode short-term memorization.

Learning associations and stimulus comparisons with local synaptic plasticity and TSP

We first tested whether the model is able to learn general associations between sensory input patterns. To this end, we generated two disjunct sets $\mathcal{R} = \{r_1, \dots, r_n\}$, $\mathcal{S} = \{s_1, \dots, s_n\}$, each consisting of n Omniglot [34] characters. The network had n output neurons, one for each element in $\mathcal{S}$ and each element in $\mathcal{S}$ was associated with one output neuron. Before each episode, we randomly drew a bijective mapping between these two sets to generate n stimulus pairs from, $\mathcal{R} \times \mathcal{S}$ such that each element from the two sets appeared in exactly one pair. These pairs were shown to the network in random order, representing facts to be memorized by the network, see Fig 3A. Afterwards, an input consisting of one element of $\mathcal{R}$ was shown as a query and the network had to activate the output neuron associated with the paired stimulus.

The network consisted of $d = 128$ input neurons in each of the apical and basal input populations, 200 neurons in the memory layer, and n output neurons. In order to generate a reasonable higher-level representation of the Omniglot characters, each character of the presented pair was first embedded in 64-dimensional space with a convolutional network pre-trained using a prototypical objective [35] (the pre-training was done on a subset of the Omniglot classes which were not used in $\mathcal{R}$ and $\mathcal{S}$). The embeddings were then concatenated in a 128-dimensional vector $\mathbf{x}^t$. Then, both the apical and basal input populations were activated with this vector, i.e., $\mathbf{x}^{a,t} = \mathbf{x}^t$ and $\mathbf{x}^{b,t} = \mathbf{x}^t$. At query time, one character from $\mathcal{R}$ was embedded and concatenated with a 64-dimensional zero vector to obtain $\mathbf{x}^{a,t}$ to provide the input to the apical compartment, while there was no input to the basal compartment ($\mathbf{x}^{b,t} = \mathbf{0}$). We trained the network using a reward-based paradigm where it received a positive reward for the correct response at query time and a negative reward otherwise. In this reward-based setting, we used the standard proximal policy optimization (PPO) objective with an entropy bonus [36] (see section 'General simulation details' in *Methods* for a definition) to compute

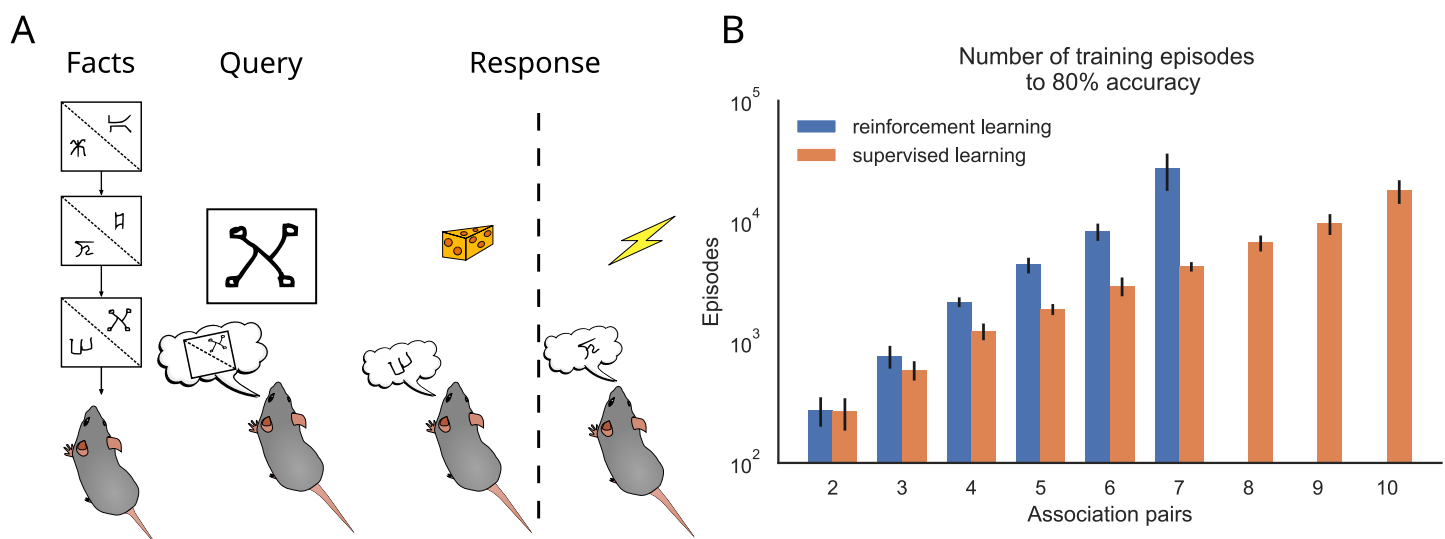


Fig 3. Learning associations with local plasticity. (A) An agent observes a sequence of stimulus pairs. After being cued by one of the observed stimuli, it has to indicate the associated one. (B) Number of training episodes needed until the network achieved an accuracy of 80% as a function of association pairs to be remembered (mean and SD over 16 training trials).

<https://doi.org/10.1371/journal.pone.0313331.g003>

the learning signals used for the synaptic weight updates. The network achieved 100% accuracy on this task after about 2000 episodes for $n = 5$ association pairs. Fig 3B shows how learning time scales with the number of associations to be learned (blue bars). A few hundred episodes suffice for two associations. For comparison, we also trained the network with direct supervision (error signals are determined from the target response; orange bars; see section 'General simulation details' in *Methods*). As expected, learning time increase is milder, but with a similar tendency. Neural network models can be sensitive to parameter settings. In order to test whether our model is robust to parameter changes, we analyzed the impact of the hyperparameters γ_+ , γ_- , as well as the memory layer size on network performance. We found that network performance is stable over a large range of these parameters. See Section B in S1 Appendix for details of the analysis.

Another memory-related task frequently used in experiments is the classical delayed match-to-sample task [37]. Here, the animal observes two stimuli separated in time and must produce an action depending on whether or not the two stimuli are equal, see Fig 4A. We modeled this task in the setup described above with a pre-trained convolutional network to embed the stimulus in 64-dimensional space and 200 neurons in the memory layer. The agent first observed one out of five Omniglot characters c_1 , followed by eight steps where white noise input is shown. Finally, another character c_2 was shown which was chosen to be the same as c_1 with probability 0.5 and one of the other characters with probability 0.125 each. The output of the network was then interpreted as an action $a \in \{left, right\}$ indicating a match or non-match. A reward was delivered accordingly which was used to compute the learning signals for synaptic weight updates. Then, another episode started.

In this task, we also tested whether the network can cope with variance in the input stimuli. In Omniglot, each character class consists of 20 drawings of the character from different

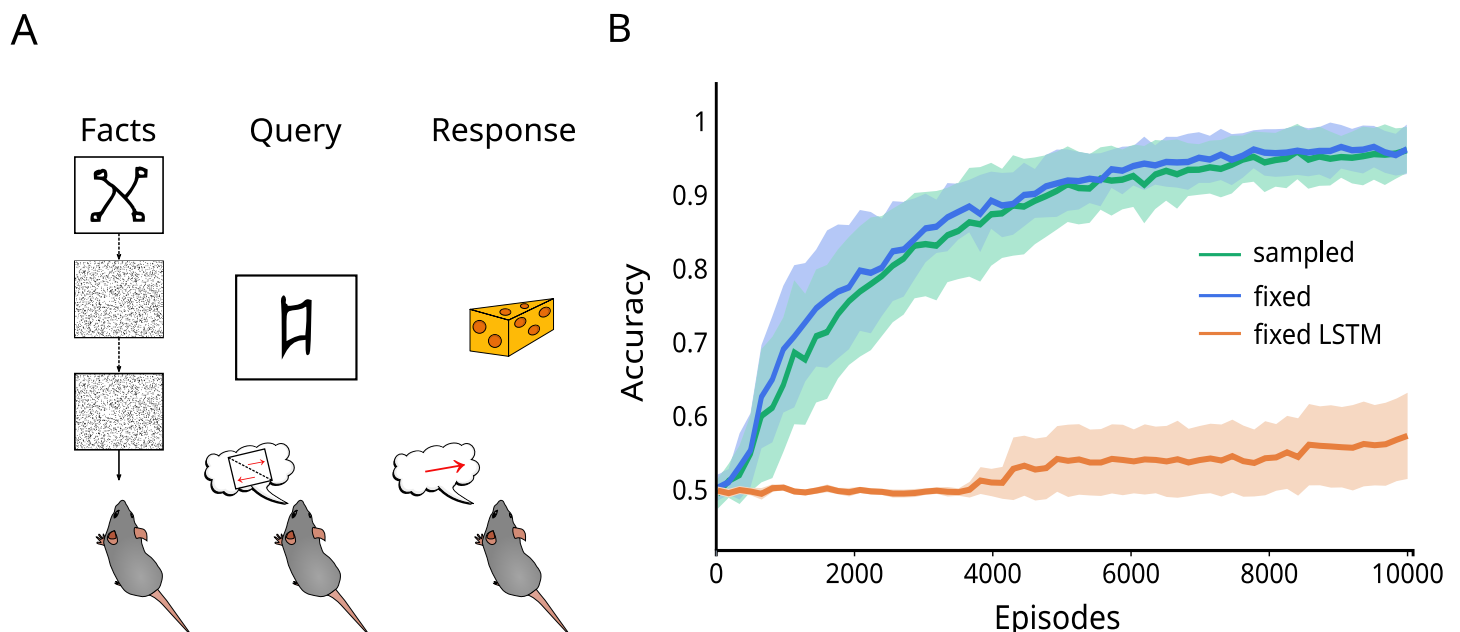


Fig 4. Learning of a delayed match-to-sample task with local plasticity. (A) Task schema. The agent observes a stimulus c_1 followed by 8 white noise inputs and another stimulus s_2 . The agent should choose the *left* action when the initial stimulus c_1 matches query stimulus c_2 . (B) Learning progress in terms of choice accuracy. Green: Only one character instantiation per class for training and testing. Blue: Network is tested on a character not seen during training. Brown: LSTM in the fixed setting (16 trials; shading indicates standard deviation)

<https://doi.org/10.1371/journal.pone.0313331.g004>

people with significant variance. We compared network performance in a setting when the specific instantiation of the presented character was fixed to a setting where it was drawn randomly from the set in each episode. In particular, for performance evaluation, we used a character instantiation that was not used for training (see *Methods* for details). Training progress is shown in Fig 4B. The network achieved an accuracy of $95 \pm 1.8\%$ on this task, with no significant difference between the fixed-character and sampled-character settings. We wondered whether the generalization capabilities of the network in the sampled-character setting could be fully contributed to a writer-independent representation of the characters in the embedding of the convolutional network. We therefore visualized the variance of embeddings for different samples of a given character using principal component analysis and t-SNE [38], see Section C in S1 Appendix. We found that there is still significant variance in these embeddings. Although the convolutional embedding certainly helps for generalization, this shows that the inputs do not need to be rigidly symbolic. Rather, the network can deal with variability in the input representation provided by the convolutional network.

We also tested the performance of a long short-term memory (LSTM) network [39] trained with BPTT, i.e., with non-local plasticity, see Fig 4B. Interestingly, the LSTM was not able to learn this task consistently (it reached performances of around 90% in some trials but failed in others). We considered one LSTM with the same number of neurons and one with the same number of parameters as our network. Fig 4B shows the better performing one. This shows that TSP improves the learning capabilities of neural networks with local synaptic plasticity on this task.

While the trunk strength can in principle hold information over arbitrary durations, the noise in the input during the delay period induces trunk strength changes which perturb the memory, leading to a drop in accuracy for increasing delays. See Section D in S1 Appendix for an analysis and discussion.

Learning context-dependent reward associations with local synaptic plasticity and TSP

We next tested whether the network trained with local synaptic plasticity rules was also able to learn a more complex radial maze task [40]. In this task, the animal is located in an eight-armed radial maze (see Fig 5). It observes one out of four context inputs (in our model, characters from the Omniglot data set), indicating one pair of arms that can be entered (indicated by color in Fig 5A). For each context, one of the two arms contains a reward. In the beginning of each episode, the branch containing the reward is randomly assigned for each of the contexts and held fixed for the episode duration. The animal has to first explore in which of the branches the reward is located and then remember this information for each of the four contexts separately throughout the episode, requiring memorization abilities of the animal. Each episode in this task consisted of 40 trials (i.e., 40 context stimuli and arm choices). In each episode, the reward locations were chosen randomly initially and stayed constant throughout that 40-trial episode. Hence, the task demands memorization of the reward location within each episode. The memorized information can then be used to choose the rewarded location in the remaining trials of the episode.

We modelled this task using a network as described above, where the visual context was embedded in a 64-dimensional vector using the same pre-trained convolutional network and a 200 neuron memory layer. At the beginning of a trial, one arm pair was chosen randomly out of the four possible pairs and the context stimulus c was presented to the network. The network output was interpreted as an action $a \in \{left, right\}$ to choose one of the available

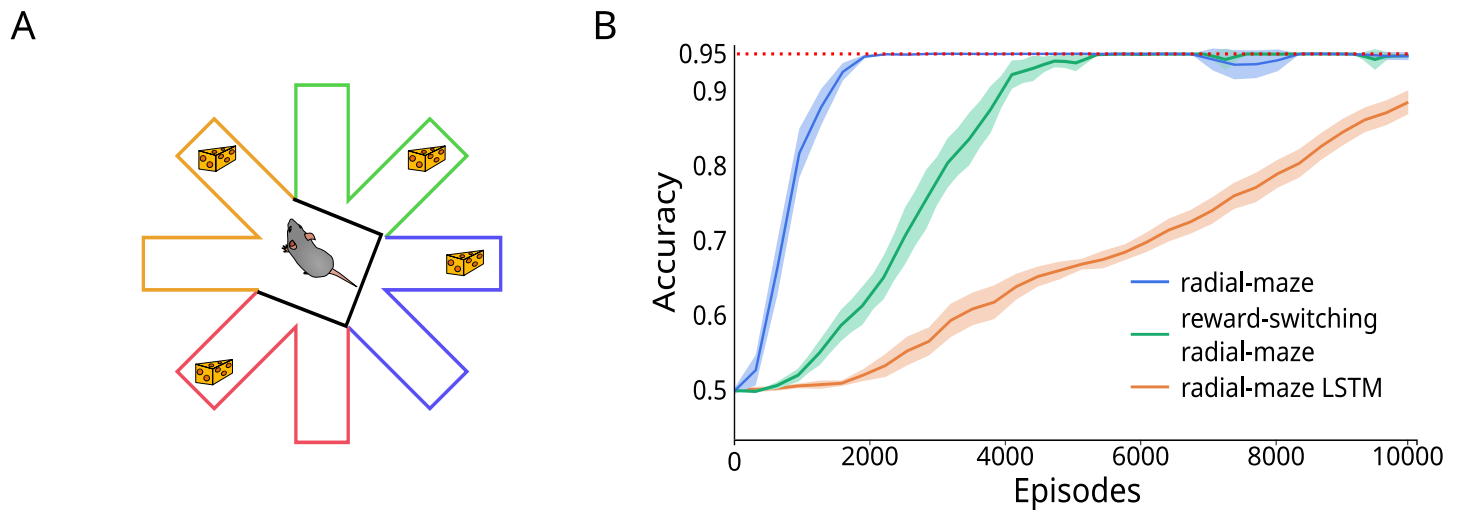


Fig 5. Context-dependent reward associations. (A) Schema of the radial maze task. In each trial, one arm pair is accessible to the agent (yellow in the example) and the context cue is presented (Omniglot character). The agent then has to choose the correct arm (*left* or *right*) to obtain the reward. (B) Fraction of rewarded actions over learning episodes in the basic radial maze task (blue) and the same task where the rewarding arm is switched after visit (green; mean and SD. over 16 runs). Red: maximum achievable performance. Orange: LSTM in the basic radial maze task.

<https://doi.org/10.1371/journal.pone.0313331.g005>

arms. The network then received either a positive or negative reward, which was used to compute learning signals and update synaptic weights. Afterwards, the network observed a summary of this trial through a triple (c, a, r) , consisting of the context stimulus c , the chosen action a , and the binary variable $r \in \{0, 1\}$ indicating the received reward. This information could be used by the network to memorize the rewarding action in the given context. Further details regarding the task setup can be found in *Methods*.

We measured the performance of the network through choice accuracy, that is, the average fraction of rewarding choices within episodes (Fig 5B). Since the agent has to guess the rewarding action per context initially, the maximum expected accuracy is 0.95. The network learned this task perfectly within about 2000 episodes. Note that this task also includes some basic form of counterfactual reasoning, since the agent can reason about the reward location when visiting a non-rewarded arm. We also tested a more complex variant of this task where the reward in the visited context switches to the other arm after the visit. Also, this task could perfectly be learned with local synaptic plasticity within approximately twice as many episodes compared to the non-switching case. An analysis of the learned network solution can be found in Section E in S1 Appendix. We also evaluated the performance of an LSTM network with BPTT on the basic version of the task, see Fig 5B. The LSTM was converging towards a solution, but learned much slower.

Learning question answering tasks with local synaptic plasticity and TSP

In the above simulations, we tested our model on standard experimental paradigms: a delayed match-to-sample task, and a radial maze task. We next asked whether local synaptic plasticity rules could learn to harness TSP to solve more complex cognitive tasks. One standard benchmark for memory-augmented neural networks is the bAbI task set [17]. It consists of 20 question-answering tasks, where each task is composed of a story consisting of a sequence of up to 325 sentences, followed by a question for which the answer can be inferred from the information contained in the story. See Section F in S1 Appendix for example tasks. For our

experiments, we used the 10k bAbI dataset to train a network with 200 neurons in the memory layer. According to the benchmark guideline, a task is considered as solved if the error rate is less or equal to 5%. Each sentence of a story was embedded in an 80-dimensional vector, and the sequence of these embeddings was presented to the network sequentially. We first considered a *random embedding*, where we generated an 80-dimensional random vector for each word using the He-uniform variance scaling method [41]. For a given sentence, the vectors of all words in the sentence were then linearly combined with a position encoding that encodes the position of the word in the sentence as in [20]. Further details regarding the task setup can be found in *Methods*.

In Table 1 we report the mean error rate of the model over 5 runs for each of the 20 bAbI tasks.

Using the random embedding, the network was able to learn 13 tasks using local synaptic plasticity (column 3). As a baseline, we considered the same network architecture trained with BPTT (column 2). Notably, all tasks for which the network could be optimized by BPTT could also be learned by our temporally and spatially local synaptic plasticity rules, showing the effectiveness of local learning in our model. Here, the model was trained separately on each of the tasks, resulting in one network for each task. We next tested whether a single network was able to solve all the tasks that could be solved by individual networks by training one model jointly on these tasks (column 4). We found that this was the case and that error rates on a majority of these tasks were even improved, indicating a knowledge transfer between them during learning. In order to test how much a more task-specific sentence representation would improve the results, we also considered a *pre-trained embedding* which is optimized for the task. Here, we used the random embedding as initialization and trained the embedding end-to-end using BPTT on the task at hand. Then the embedding was fixed,

Table 1. Comparison of mean errors of a network trained with BPTT vs our local synaptic plasticity on bAbI tasks 10k (mean and SD. over 5 trials). Error rates for tasks solved by using our local synaptic plasticity rules are printed in bold face. BPTT: backpropagation through time; LSP: Local synaptic plasticity; LSP joint: joint training where a single network was trained to perform all tasks concurrently.

Tasks	Random embedding			Pre-trained embedding	
	BPTT	LSP	LSP joint	BPTT	LSP
1: Single Supporting Fact	0.3 ± 0.1	0.6 ± 0.2	0.2 ± 0.1		
2: Two Supporting Facts	55.6 ± 0.9	57.8 ± 0.9		35.7 ± 0.4	37.4 ± 0.2
3: Three Supporting Facts	58.4 ± 0.7	62.4 ± 1.1		48.2 ± 0.3	48.8 ± 0.5
4: Two Arg. Relations	0.02 ± 0.1	0.06 ± 0.1	0.0 ± 0.0		
5: Three Arg. Relations	1.4 ± 0.2	1.8 ± 0.2	2.3 ± 0.1		
6: Yes/No Questions	1.0 ± 0.3	1.4 ± 0.2	2.0 ± 0.1		
7: Counting	1.5 ± 0.2	2.6 ± 0.4	1.4 ± 0.2		
8: Lists/Sets	1.5 ± 0.3	0.9 ± 0.2	2.3 ± 0.1		
9: Simple Negation	2.3 ± 0.2	2.4 ± 0.2	0.3 ± 0.1		
10: Indefinite Knowledge	3.3 ± 0.1	3.2 ± 0.1	3.2 ± 0.3		
11: Basic Coreference	3.3 ± 0.2	3.1 ± 0.2	3.0 ± 0.1		
12: Conjunction	2.1 ± 0.2	2.7 ± 0.2	0.4 ± 0.1		
13: Compound Coref.	7.0 ± 0.5	7.5 ± 0.5		2.7 ± 0.1	2.9 ± 0.2
14: Time Reasoning	15 ± 0.8	14.5 ± 0.8		10.1 ± 0.2	11.8 ± 0.3
15: Basic Deduction	0.6 ± 0.2	1.0 ± 0.4	0.8 ± 0.3		
16: Basic Induction	48 ± 10	51.3 ± 2.1		38.8 ± 3.6	47.8 ± 2.4
17: Positional Reasoning	19 ± 0.5	17.4 ± 1.0		3.7 ± 0.5	4.7 ± 0.5
18: Size Reasoning	4.7 ± 0.1	4.8 ± 0.3	2.6 ± 0.4		
19: Path Finding	56.3 ± 0.5	61.3 ± 0.9		1.9 ± 0.1	2.6 ± 0.3
20: Agent's Motivations	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0		

<https://doi.org/10.1371/journal.pone.0313331.t001>

and a fresh network was trained with local synaptic plasticity on this input representation. We tested all tasks that were not solved with the random embedding, and found that three additional tasks could be solved with a better input representation (column 6). Again, all tasks that could be solved with BPTT could also be learned with local synaptic plasticity.

Overlapping assembly representations emerge through local learning

How does the network solve such tasks after training? In order to answer this question, we analyzed the behavior of a trained network in the “Single supporting fact” task, see Fig 6. This task involves simple person-location relations such as “John moved to the kitchen” among several persons and possible locations. Although the results are presented with the verb “moved”, we note that there are actually various phrases in the story to indicate such person-location relations. Those variations do not alter the person-location relation, and the model should thus learn to treat them as equal. To analyze stories, we recorded the vectors of trunk strengths $\mathbf{h}^f$ that arose from the dataset’s stories. These vectors thus represent the memory state of the memory layer throughout the stories. Subsequently, we applied a non-negative matrix factorization (NMF) [42] to project these memory states into a 20-dimensional space. We then projected the memory states, keys, values, and recall keys into this space.

Fig 6A (top) shows the average activity in the key layer for a given person, where the average was taken over all possible locations and action verbs. For example, the leftmost visualized vector shows sentences of the form “john moved to the ...”, where the location is marginalized out. When comparing the representations for four persons, one observes that keys effectively discriminate between persons with close to orthogonal representations. When we performed the same analysis for a given location, averaging over persons, we did not observe such a structure (not shown). In contrast, such orthogonal representations can be observed for specific locations in the value layer when an average was taken over persons and action values (Fig 6A, middle). We found that the orthogonality of representations depends on the memory size and is crucial for task performance, see Section G in S1 Appendix.

In summary, we observed that keys effectively discriminate between persons, while values indicate locations. We thus define each person’s key vector as the average of the key vector projections from all possible variations of sentences for this person. Similarly, we define a location’s value vector as the average of the value vector projections from all possible variations of sentences for this location.

While key- and value-representations are orthogonal to those of the same layer, we found a systematic overlap between key representations with value representations. For instance, John’s key vector primarily overlaps with specific locations indicated by the value vectors (components 1, 4, 7, and 14 overlap office, kitchen, bedroom, and hallway respectively, Fig 6A, compare top and middle). This analysis can be similarly conducted based on the value vectors. For example, the most activated components of the office’s value overlap primarily with specific people indicated by the keys (component 1, 15, 17, and 19 overlap with John, Mary, Daniel, and Sandra respectively). The bottom row of Fig 6A shows recall keys during queries for the location of persons (e.g., “where is john”). We observe that these recall keys are very similar to keys during storage operations for facts that include the same person.

This representation, that has been learned through local synaptic plasticity in our model, can be used to store relevant information in the trunk strengths of the pyramidal neurons of the memory layer. This is illustrated in Fig 6B where we illustrate the processing of a simple story. In the first sentence “Mary moved to the bedroom”, the overlap between the key- and value-representations potentiates the trunk strength of the corresponding pyramidal neurons (component 3 in our projection), which can be observed when projecting the memory state $\mathbf{h}^f$

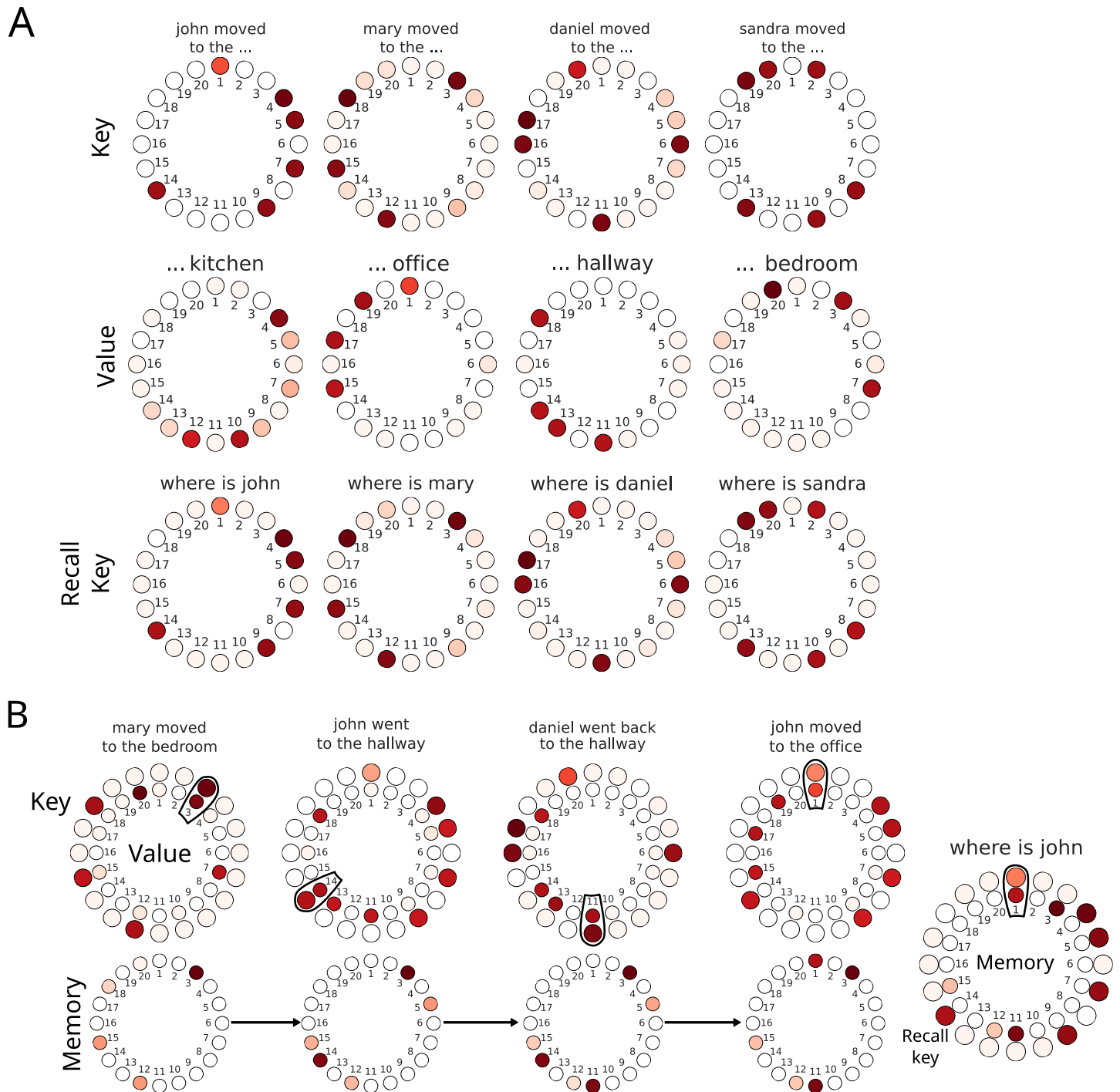


Fig 6. Network analysis for the Single Supporting Fact task. (A) Projection of keys, values, and recall keys based on a non-negative matrix factorization of memory traces after network training. Keys are shown for specific persons, with representations averaged over locations and verbs. The key for John clearly activates 6 components corresponding to possible locations for John. Value representations are shown for specific locations, with representations averaged over persons and verbs. (B) Story sample along with its respective key (top, outer ring), value (top, inner ring), and memory state after memorization (bottom). Each key and value pair predominantly overlaps in a single component, which is then memorized. Additionally, the change in John's location in the last fact is accurately updated from component 14 to 1, causing component 14 to be deactivated due to the negative term in our Oja-type Hebbian rule.

<https://doi.org/10.1371/journal.pone.0313331.g006>

after the Hebbian update into the low-dimensional space (leftmost bottom representation in Fig 6B). This potentiated state is retained after the next two sentences, and new memories are added according to the presented facts. At the last presented fact “john moved to the office”, John changes his location from the hallway to the office. This change is accurately recorded in the memory: the overlapping of component 1 in the key and value represents the new location, while the deactivation of components 14, corresponding to the previous location, occurs due to the negative term in the Hebbian rule. The final state of the memory is then combined with a specific recall key from the question, “Where is John?” The answer is determined by the overlapping between their activated components, effectively corresponding to the newly activated component 1, which is part of the office representation – John’s last change of location. Hence, the readout can easily determine office as the correct answer.

In summary, the model has learned assembly representations for entities. These representations are partly orthogonal and partly overlapping. An overlap defines a potential association that can be stored in the neurons of the memory layer. Experimental studies in humans have found clear evidence for assembly representations of celebrities and popular places in the medial temporal lobe of humans with partial overlap [43,44]. According to our model, overlapping assemblies emerge through learning because they are needed for the storage of associations in the memory layer.

Discussion

We propose in this article that memory-dependent neural processing is jointly shaped by non-synaptic and synaptic plasticity. The interaction between non-synaptic intrinsic and synaptic plasticity has been studied in previous work. These works however considered intrinsic plasticity as a homeostatic mechanism to maximize information transfer [45,46], to support unsupervised learning [15,16], or for associative learning [13]. The cooperation of intrinsic and synaptic plasticity has also been studied for information theoretic learning in artificial neural networks [47]. Another aspect of intrinsic plasticity is that it can regulate the dynamical behavior of recurrent network models [48–50]. In contrast to these works, our model shows that non-synaptic plasticity can be used as a memory buffer that is utilized by synaptic plasticity processes. Local plasticity for two-compartment neurons has been studied in [51], but in the context of representation learning.

The idea that fast synaptic plasticity can underlie working memory has been proposed already in [52], see [53] for a recent review. This idea has been extensively used in memory-augmented neural networks [20,21,29,30,54–56]. Here, however, synaptic plasticity was used as a fast memory buffer and networks were trained with backpropagation through time. Fast synaptic Hebbian plasticity was combined with reward-based learning in [57] to learn a navigation task. From the neuroscience perspective, non-synaptic fast plasticity is consistent with the observation that non-synaptic plasticity can act on a fast timescale [6,9,10]. But what could be functional advantages of non-synaptic neuron-specific plasticity? Our study indicates one potential advantage: local learning rules for synaptic connections can be used to shape the circuitry around the memory units.

The proposed network model could be improved in several directions. While we were able to show that local learning works very well when compared with BPTT (see e.g. Table 1), the simple memory architecture is limiting. Hence, when compared to H-Mem [21], a brain-inspired memory network without local learning, it fails on a number of bAbI tasks which can be solved by H-Mem. The fast memory in models such as H-Mem is implemented by a fully connected synaptic weight matrix between a neuron layer that represents the key vector and a neuron layer that represents the value vector. This more complex memory architecture

prevents the derivation of local plasticity rules with the method utilized in this article. It is an interesting question whether biologically plausible learning mechanisms could be derived for this case as well. Since Hebbian branch plasticity similar to Eq (4) was found in oblique dendrites of pyramidal cells, it would be interesting to investigate models where each oblique branch acts as a memory unit similar to our h_i^t . This could significantly boost the memory capacity of the model. Another limitation of our model is that learning can take long. In the pattern association task (Fig 3), learning of associations of two pattern-pairs needs a few hundred episodes, which increases with the number of associated pairs (Fig 3B). Here, curriculum setups and more specialized circuit architectures are potential candidates for improvement. We emphasize however that unseen associations between previously observed patterns can be memorized instantly after training by our model.

One basic assumption of our model is that the coupling strength between the apical dendrites and the soma can be modified by sub-cellular processes. Local plasticity of excitability of oblique dendrites in hippocampal pyramidal cells has been reported in [26]. The regulation of the somatic coupling of apical dendrites — to which we refer to as trunk strength plasticity in this article — could be implemented through the regulation of hyperpolarization-activated channels, which generate the I_h current [23]. It has been shown that these channels strongly influence the dendro-somatic coupling of apical dendrites [27]. Other potential mechanisms include sodium-, calcium-, or potassium-conductances to facilitate calcium spike generation. These mechanisms would result in a change of calcium-dynamics at specific locations at the pyramidal cell through learning, which could be observed with imaging techniques in behaving animals.

Our derived learning rules can be classified as extended three-factor learning rules with eligibility traces [33]. Three factor rules are implementing learning rules of the form

$$\Delta w_j = \eta e_j^t M^{3rd,t}, \quad (14)$$

$$e_j^t = \alpha e_j^{t-1} + \beta f(x_j^t) g(y^t), \quad (15)$$

where e_j is the eligibility trace for synapse j , $M^{3rd,t}$ is the third factor, α and β are constants, x_j is the presynaptic input, y^t is the postsynaptic activity, and f and g are functions specific to the considered three factor rule [33]. Typically, the eligibility trace decays (since $0 < \alpha < 1$), and is increased by coincident pre- and postsynaptic activity. There is experimental evidence for such rules in various brain areas such as striatum [58], cortex [59], and hippocampus [60–62]. One prominent three-factor rule with strong experimental support is the synaptic tagging-and-capture hypothesis [63]. There, a synapse is tagged by joint pre- and postsynaptic activity, and consolidation occurs up to an hour later via a third factor that is a neuromodulatory input. Third-factor rules are also quite often used in models for reward-based learning [64–67]. Our rules necessarily extend these rules since we have apical and basal postsynaptic activations which enter the rule, whereas the standard three-factor formulation is based on point neuron models. We would like to compare this general framework to our plasticity rule for basal synapses. To simplify the discussion, we re-formulate the rule by neglecting the impact of h and the homeostatic term in Eqs (8) and (10)–(12) to obtain

$$\Delta w_j^{\text{basal}} = \eta L^t a^t e_j^{\text{basal},t}, \quad (16)$$

$$e_j^{\text{basal},t} = e_j^{\text{basal},t-1} (1 - \gamma_+ b^t a^t) + \gamma_+ H(\bar{b}^t) a^t x_j^{\text{basal},t}. \quad (17)$$

Comparing Eq (16) to Eq (14), we see a similar structure with L^t taking the role of the third factor. In addition, weight changes are modulated by apical activation. This is reminiscent of

behavioral timescale plasticity [62], which can also be interpreted as a three factor rule where the calcium plateau potential (complex spike) is the third factor [33]. Eqs (17) and (15) also have a very similar structure. We can however observe two differences. First, eligibility traces do not decay exponentially, rather the decay depends on coincident basal and apical activity. Second, the eligibility trace is increased in (17) when presynaptic input is paired with basal and apical activity. Hence, in general the rules fit well into the three-factor framework, with an extension that takes the separation of the basal and apical compartment into account.

While the derived eligibility traces satisfy temporal locality and the utilized signals are local to the neurons, those traces depend on these signals in a complex manner. In particular, g^{basal} , g^{apical} , and f depend respectively on both the apical a^t and basal b^t activities. Although not both of these signals are directly local to the respective synapses,

evidence for bi-directional communication between the soma and distal apical dendrites has been observed [68,69], a mechanism that is thought to support coincidence detection between projected feed-forward basal and feed-back apical inputs. Such coincidence-based plasticity appears both in g^{basal} and g^{apical} as potentiation of the synapse for coincident activation of a^t and b^t . While it is unlikely that exact implementations of our proposed synaptic plasticity rules exist in the brain, it would be interesting to investigate to what degree biological plasticity processes overlap with the synaptic plasticity rules proposed in this work.

In our model, a fact is processed such that a key representation is provided to the apical dendrites of pyramidal cells in the memory layer and a value representation is provided to the basal compartments. Experimental evidence suggests that the apical region of cortical pyramidal cells receives top-down input while sensory feed-forward information arrives at the basal region [70–73]. From this perspective, keys in our model could be interpreted as a higher-level abstracted conceptual representation of the associated information, whereas values represent the lower-level associated stimuli. Upon recall, we only activate the apical dendrites. Hence, from this perspective, a query in our model is composed of higher-level conceptual representations that recalls a more lower-level representation. We note however that the distinction between bottom-up and top-down input is less clear in hippocampal networks.

We have found that task-dependent learning can lead to overlapping assembly representations, see Fig 6 that is reminiscent of representations in the human medial temporal lobe [43,44]. More recent work has shown that memories of temporally close events tend to be stored in more overlapping neural assemblies than memories of temporally more distant events [74]. This effect was examined in a modeling study [75]. Our model was not designed to generate time-distance dependent assembly overlap. The overlap that we observe in Fig 6 was learned by the model since it has a functional role for this specific task. Nevertheless, we performed additional simulations and did observe a function-independent overlap that reduces for longer temporal distances, see Section H in S1 Appendix.

In summary, we have shown that non-synaptic plasticity enables memory-dependent learning with local synaptic plasticity rules. The resulting learning architecture is quite powerful, leading to results comparable to BPTT on bAbI. To the best of our knowledge, our model is the first network that is able to learn bAbI tasks with local learning rules. The involvement of non-synaptic plasticity in memory formation has been demonstrated experimentally [6,76]. Our model proposes a functional role for it in synergy with synaptic plasticity.

Materials and methods

In this section, we provide detailed information about the model and derive the local synaptic plasticity rules.

Memory network model

Our network model consists of $2d$ input neurons separated into apical and basal populations with activities $\mathbf{x}^{a,t}$ and $\mathbf{x}^{b,t}$ respectively. The input neurons project to apical and basal compartments of m memory neurons in a fully-connected manner via weight matrices W^{apical} and W^{basal} respectively. For the i -th neuron in this memory layer, the resulting basal activation b_i^t and apical activation a_i^t are thus given by:

$$b_i^t = \sigma(\bar{b}_i^t), \quad a_i^t = \sigma(\bar{a}_i^t), \quad (18)$$

with the rectified linear nonlinearity, $\sigma(s) = \max\{0, s\}$, and pre-activations

$$\bar{b}_i^t = \sum_j w_{ij}^{\text{basal}} x_j^{b,t}, \quad \bar{a}_i^t = \sum_j w_{ij}^{\text{apical}} x_j^{a,t}. \quad (19)$$

with $\mathbf{x}^{a,t}$ and $\mathbf{x}^{b,t}$ as apical and basal input vectors respectively, at time step t . We included a bias in each neuron by providing a constant input $x_d^{b,t} = x_d^{a,t} = 1$. In this way, the derivation of the update rule for the bias can be performed together with the synaptic weights. In the resulting rules, only the input has to be set to 1 in order to obtain the simplified bias updates. The output z_i^t of each cell is then given by a linear combination of these two activations, whereof a_i^t is multiplied with the trunk excitability h_i^t :

$$z_i^t = b_i^t + h_i^t a_i^t. \quad (20)$$

The trunk excitability h_i^t is updated according to the Hebbian update

$$\begin{aligned} \Delta h^t &= \gamma_+ (h^{\max} - h^{t-1}) b^t a^t - \gamma_- h^{t-1} (a^t)^2, \\ \bar{h}^t &= h^{t-1} + \Delta h^t, \\ h^t &= \sigma(\bar{h}^t), \end{aligned} \quad (21)$$

with σ the ReLU non-linearity, parameters $\gamma_+, \gamma_- > 0$, and $h^{\max} > 0$ is the maximum trunk strength.

The network output was then computed as

$$y_k^t = \sum_i w_{ki}^{\text{out}} z_i^t, \quad (22)$$

for output weight matrix W^{out} , where $\mathbf{z}^t$ is determined by (20). Note, that in the reinforcement learning case, we further split this readout weight matrix into $\mathbf{w}_V$ and W_π , respectively, the value and policy weights, but here this separation is omitted to simplify notation. For these cases, W^{out} can be interpreted as a concatenation of $\mathbf{w}_V$ and W_π into a single matrix.

We distinguish between a *memorization event*, where the memory is updated according to Eq (21) and the model output is discarded, and a *memory recall event*, where the model produces an output $\mathbf{z}^t$ based on the memory state $\mathbf{h}^t$ as in Eq (20), but no memory update is performed. During memory recall events, only the apical input population is active, the basal input population is defined to be silent, i.e. $\mathbf{x}^{b,t} = \mathbf{0}$. In the reinforcement learning setup, we execute a memorization event followed by a memory recall event at every single time step. The network output of the recall event is then used to update the state value estimator. In the supervised learning setup, network outputs are only needed when an action is demanded (we

call this a *query step* and the input at this step a *query*). Therefore, we perform memory recall events at query steps and memorization events are performed for other time steps.

Derivation of local synaptic plasticity rules

Here, we derive gradient-based local plasticity rules for the slow synaptic weights W^{out} in the output layer and $W^{\text{apical}}, W^{\text{basal}}$ in the memory layer (in contrast to the fast trunk strengths in the memory layer neurons).

In vanilla backpropagation through time (as commonly used for training recurrent networks), symmetrical feedback connections are used to back-propagate the error gradients through the unrolled network. These symmetrical feedback weights are generally considered biologically implausible due to the weight transport problem [77]. In this work we use the e-prop framework [32] to derive local learning rules for every synapse, that require neither the application of symmetric weights, nor the propagation of errors to preceding time steps.

Output layer. For the readout layer weights W^{out} , local learning signals are already available because the output of this layer is directly used to obtain the error signal E . We can therefore update output weight w_{ki}^{out} from memory layer neuron i to output neuron k with

$$\Delta w_{ki}^{\text{out}} = -\eta \frac{\partial E}{\partial w_{ki}^{\text{out}}} = -\eta \frac{\partial E}{\partial y_k^t} \frac{\partial y_k^t}{\partial w_{ki}^{\text{out}}} = -\eta \frac{\partial E}{\partial y_k^t} z_i^t, \quad (23)$$

where $\eta > 0$ denotes the learning rate.

Basic e-prop framework and feedback alignment. To perform gradient descent on weights in the memory layer, we need to compute the gradients

$$\frac{dE}{dw_{ij}^\theta} = \sum_t \frac{dE}{dz_i^t} \cdot \frac{dz_i^t}{dw_{ij}^\theta}, \quad (24)$$

with $\theta \in \{\text{apical}, \text{basal}\}$. The second term $\frac{dz_i^t}{dw_{ij}^\theta}$ can be expressed as an eligibility trace, see below. The first term $\frac{dE}{dz_i^t}$ however necessitates the backpropagation of error signals through time. One core idea of the e-prop formalism [32] is to replace this gradient by a temporally local approximation,

$$\frac{dE}{dz_i^t} \approx \sum_k \frac{\partial E}{\partial y_k^t} \frac{\partial y_k^t}{\partial z_i^t} = \sum_k \frac{\partial E}{\partial y_k^t} w_{ki}^{\text{out}}, \quad (25)$$

$$\approx \sum_k \frac{\partial E}{\partial y_k^t} b_{ik}^{\text{fb}} \stackrel{\text{def}}{=} L_i^t, \quad (26)$$

where index k denotes the k -th neuron in the output layer. Here, L_i^t is interpreted as a neuron-specific learning signal for neuron i in the memory layer.

In the above equations, we distinguish between the total derivative $\frac{df}{dy}$ and the partial derivative $\frac{\partial f}{\partial y}$. The function f may depend on many variables, where many of them may also depend on y . The total derivative takes all these dependencies into account, while in the partial derivative there is only a direct dependency of f on y . More details about this notation can be found in [32], where the e-prop formalism was proposed.

The approximation in Eq (26) implements feedback alignment, where the symmetric feedback weights are replaced by randomly chosen weights [78]. In our simulations, we used the adaptive e-prop method [32], where the feedback weights B^{fb} are initialized randomly

and then undergo the same weight updates as weights in W^{out} : $\Delta b_{ik}^{\text{fb}} = \Delta w_{ki}^{\text{out}} = \frac{\partial E}{\partial w_{ki}^{\text{out}}}$. In addition, these two weight matrices are subject to L2 weight decay. This ensures, that after some iterations W^{out} and B^{fb} will converge to similar values.

Memory layer. By making use of the learning signals L_i^t from the previous section, we can find local synaptic plasticity rules to update the synaptic weights W^{apical} and W^{basal} of the apical and basal compartment respectively. We derive local gradients $\frac{dE}{dw_{ij}^{\text{apical}}}$ and $\frac{dE}{dw_{ij}^{\text{basal}}}$ to minimize the error E through the e-prop formalism [32]. With these local gradients, it is not necessary to back-propagate the error signal through all time steps of the computation. Instead, eligibility traces are forward-propagated and used in conjunction with learning signals to determine the gradient required for the weight updates.

Apical compartment. To find the derivative $\frac{dE}{dw_{ij}^{\text{apical}}}$ of the error function E with respect to apical weight w_{ij}^{apical} , we factorize

$$\frac{dE}{dw_{ij}^{\text{apical}}} = \sum_t \frac{dE}{dz_i^t} \cdot \frac{dz_i^t}{dw_{ij}^{\text{apical}}} . \quad (27)$$

Now we use our learning signal L_i^t to approximate $\frac{dE}{dz_i^t}$, which is a core idea in the e-prop formalism [32]. We also expand $\frac{dz_i^t}{dw_{ij}^{\text{apical}}}$ by applying the product rule, resulting in

$$\frac{dE}{dw_{ij}^{\text{apical}}} \approx \sum_t L_i^t \cdot (a_i^t \frac{dh_i^t}{dw_{ij}^{\text{apical}}} + \frac{\partial a_i^t}{\partial w_{ij}^{\text{apical}}} h_i^t) . \quad (28)$$

Unrolling $\frac{dh_i^t}{dw_{ij}^{\text{apical}}}$ in time, we obtain

$$\frac{dh_i^t}{dw_{ij}^{\text{apical}}} = \sum_{l=1}^{t-1} \left(\prod_{k=l+1}^t \frac{\partial h_i^k}{\partial h_i^{k-1}} \right) \frac{\partial h_i^l}{\partial w_{ij}^{\text{apical}}} + \frac{\partial h_i^t}{\partial w_{ij}^{\text{apical}}} . \quad (29)$$

We can re-write this definition as a recursive function and hereby introduce our eligibility trace $e_{ij}^{\text{apical},t}$:

$$\frac{dh_i^t}{dw_{ij}^{\text{apical}}} = e_{ij}^{\text{apical},t} = \frac{\partial h_i^t}{\partial h_i^{t-1}} e_{ij}^{\text{apical},t-1} + \frac{\partial h_i^t}{\partial w_{ij}^{\text{apical}}} . \quad (30)$$

We observe, that this recurrent definition only depends on partial derivatives ($\frac{\partial h_i^t}{\partial h_i^{t-1}}$ and $\frac{\partial h_i^t}{\partial w_{ij}^{\text{apical}}}$) that are local in time and can therefore be easily derived from Eq (21) as:

$$\frac{\partial h_i^t}{\partial h_i^{t-1}} = H(\bar{h}_i^t) (1 - \gamma_+ b_i^t a_i^t - \gamma_- (a_i^t)^2) , \quad (31)$$

$$\frac{\partial h_i^t}{\partial w_{ij}^{\text{apical}}} = H(\bar{h}_i^t) H(\bar{a}_i^t) \gamma_+ [(h^{\text{max}} - h_i^{t-1}) b_i^t - 2\gamma_- h_i^{t-1} a_i^t] x_j^{a,t} , \quad (32)$$

where $H(x)$ is the Heaviside step function used as derivative of the ReLU function.

By plugging Eqs (31) and (32) into (30), we obtain:

$$e_{ij}^{\text{apical},t} = e_{ij}^{\text{apical},t-1} f(h_i^t, b_i^t, a_i^t) + g^{\text{apical}}(h_i^t, h_i^{t-1}, b_i^t, a_i^t) x_j^{\text{a},t}, \quad (33)$$

with

$$f(h_i^t, b_i^t, a_i^t) = H(\bar{h}_i^t)(1 - \gamma_+ b_i^t a_i^t - \gamma_- (a_i^t)^2), \quad (34)$$

$$g^{\text{apical}}(h_i^t, h_i^{t-1}, b_i^t, a_i^t) = H(\bar{h}_i^t) H(\bar{a}_i^t) \gamma_+ [(h^{\text{max}} - h_i^{t-1}) b_i^t - 2\gamma_- h_i^{t-1} a_i^t], \quad (35)$$

as stated in the main text. We finally use this eligibility trace to substitute $\frac{dh_i^t}{dw_{ij}^{\text{apical}}}$ in Eq (28):

$$\frac{dE}{dw_{ij}^{\text{apical}}} \approx \sum_t L_i^t \cdot (a_i^t e_{ij}^{\text{apical},t} + H(a_i^t) x_j^{\text{a},t} h_i^t). \quad (36)$$

where the second term is the derivative $\frac{\partial a_i^t}{\partial w_{ij}^{\text{apical}}} = H(a_i^t) x_j^{\text{a},t}$ of our apical activation a_i^t with respect to the apical weight w_{ij}^{apical} .

Basal compartment. Analogously, we compute the derivative of the error E with respect to our basal weight matrix W^{basal} . We again start off by inserting our learning signal L_i^t :

$$\frac{dE}{dw_{ij}^{\text{basal}}} = \sum_t \frac{dE}{dz_i^t} \cdot \frac{dz_i^t}{dw_{ij}^{\text{basal}}}, \quad (37)$$

$$\approx \sum_t L_i^t \left(\frac{db_i^t}{dw_{ij}^{\text{basal}}} + a_i^t \frac{dh_i^t}{dw_{ij}^{\text{basal}}} \right). \quad (38)$$

Because of our assumptions, L^t is only non-zero during recall events and the basal compartment does not receive any inputs at this point (i.e. $x_j^{\text{b},t} = 0$), then $\frac{db_i^t}{dw_{ij}^{\text{basal}}} = 0$. Analogous to Eq (29), we can also express $\frac{dh_i^t}{dw_{ij}^{\text{basal}}}$ recurrently via eligibility trace $e_{ij}^{\text{basal},t}$:

$$\frac{dh_i^t}{dw_{ij}^{\text{basal}}} = e_{ij}^{\text{basal},t} = \frac{\partial h_i^t}{\partial h_i^{t-1}} e_{ij}^{\text{basal},t-1} + \frac{\partial h_i^t}{\partial w_{ij}^{\text{basal}}}, \quad (39)$$

where the local derivative $\frac{\partial h_i^t}{\partial h_i^{t-1}}$ was already defined in Eq (31). To obtain $\frac{\partial h_i^t}{\partial w_{ij}^{\text{basal}}}$ we can simply compute the derivative of Eq (21) w.r.t. w_{ij}^{basal} :

$$\frac{\partial h_i^t}{\partial w_{ij}^{\text{basal}}} = H(h_i^t) H(b_i^t) \gamma_+ (h^{\text{max}} - h_i^{t-1}) a_i^t x_j^{\text{b},t}. \quad (40)$$

Inserting this back into (39) yields:

$$e_{ij}^{\text{basal},t} = e_{ij}^{\text{basal},t-1} f(h_i^t, b_i^t, a_i^t) + g^{\text{basal}}(h_i^t, h_i^{t-1}, b_i^t, a_i^t) x_j^{\text{b},t},$$

with $f(h_i^t, b_i^t, a_i^t)$ equivalent to Eq (34) and

$$g^{\text{basal}}(h_i^t, h_i^{t-1}, b_i^t, a_i^t) = H(\bar{h}_i^t) H(\bar{b}_i^t) \gamma_+ (h^{\text{max}} - h_i^{t-1}) a_i^t. \quad (41)$$

Taken together, our approximated derivative reads:

$$\frac{dE}{dw_{ij}^{\text{basal}}} \approx \sum_t L_i^t \cdot a_i^t e_{ij}^{\text{basal},t}. \quad (42)$$

Weight updates in simulations

The above derived approximate gradients were accumulated over mini-batches of training examples and used for parameter updates in our model. In practice, we used these approximate gradients in the Adam [79] algorithm, which amounts to a synapse-specific learning rate based on global learning rate λ_{Adam} along with a momentum update. We used the default values for other hyperparameters of Adam which are $\beta_{1,Adam} = 0.9$, $\beta_{2,Adam} = 0.999$ and $\epsilon_{Adam} = 1e - 07$. We also used an L2 regularization term for each of the synaptic weight matrices. The hyperparameters for learning are specified in subsection 'Hyperparameters' below. For efficiency reasons, we accumulate the synaptic weight changes over mini-batches of episodes before applying them, which is a common practice in neural network training.

Temporal normalization of input neuron activity

In order to stabilize training, activity normalization techniques such as batch normalization and layer normalization are often used in artificial neural networks. In biological networks, such normalization may be carried out through inhibitory networks. In contrast to standard techniques, we have to ensure in our model that no information can be inferred from the future, therefore normalization needs to be performed in an online manner. We normalized input apical and basal activity vectors $\mathbf{x}^{a,t}$ and $\mathbf{x}^{b,t}$ component-wise over the temporal dimension t . In our case, we use a variance-mean online computation based on [80]:

$$\mu_j^t = \mu_j^{t-1} + \frac{\kappa_j^t - \mu_j^{t-1}}{t} \quad (43)$$

$$m_j^t = m_j^{t-1} + (\kappa_j^t - \mu_j^{t-1}) \cdot (\kappa_j^t - \mu_j^t) \quad (44)$$

$$\bar{\kappa}_j^t = \begin{cases} \kappa_j^t & \text{if } t < 2 \\ \frac{\kappa_j^t - \mu_j^t}{\sqrt{\frac{m_j^t}{t-1} + \epsilon}} & \text{otherwise} \end{cases} \quad (45)$$

with $\kappa^t \in \{\mathbf{x}^{a,t}, \mathbf{x}^{b,t}\}$ at time t , online approximated mean μ^t and cumulative squared difference $\mathbf{m}_j^t$ between input and mean. We further define $\mu^0 = \mathbf{m}^0 = \mathbf{0}$.

For each task, we calculated μ^t and $\mathbf{m}^t$ online during the training phase across all training samples to obtain μ_{train} and $\mathbf{m}_{\text{train}}$ to apply them as constants for the normalization during inference. This procedure is a common practice and coherent with standard normalization techniques like Batch Normalization [81].

General simulation details

Hyperparameter decay. We applied an exponential decay for some of the hyperparameters. The decay function of a hyperparameter Θ was defined as

$$\Theta^t = \begin{cases} \Theta_0 & t < \tau_\Theta \\ \Theta_0 \lambda_\Theta^{(t-\tau_\Theta)} & \text{otherwise} \end{cases} \quad (46)$$

with constant decay rate $\lambda_\Theta < 1$ and delay τ_Θ . This decay was used for the general learning rate λ_{adam} of the ADAM optimizer, the entropy coefficient ξ used in the reinforcement learning setups (these parameters are described below), and Hebbian parameters γ_+ and γ_- .

Reinforcement learning. For the reinforcement learning experiments, we used Proximal Policy Optimization (PPO) [82] with an entropy bonus [83]. For this setup, output layer weights W^{out} were split into $w_V \in \mathbb{R}^{|z|}$ and $W_\pi \in \mathbb{R}^{|z| \times l}$ with l different available actions. w_V is used to compute a scalar state value estimate $V(s^t) = \sum_i w_{V,i} z_i^t$ and W_π to compute the policy vector $\vec{\pi}^t = \text{softmax}(W_\pi z^t)$ from network output z^t . The policy vector $\vec{\pi}^t \in \mathbb{R}^l$ represents a probability distribution over all possible actions l at time step t . To train both outputs jointly, we defined an error function consisting of an additive combination of a policy error E_π and state value estimate error E_V . This corresponded to an actor-critic setup, where the network learned to represent both actor (via policy $\vec{\pi}(a^t|s^t)$) and critic (via state value estimate $V(s^t)$).

The overall error E could therefore be defined as

$$E = E_V + E_\pi, \quad (47)$$

with state value error E_V and policy error E_π .

State value error. The error for our state value estimate $V(s^t)$ was computed by using the temporal difference (TD) error [84]:

$$E_V(s, \theta) = \|r_{t+1} + \gamma \bar{V}_\theta(s^{t+1}) - V_\theta(s^t)\|_2^2, \quad (48)$$

with reward r_{t+1} , a constant γ and model parameters θ . $\|\cdot\|_2^2$ denotes the squared L2 norm. The notation $\bar{V}$ is used to indicate that this term is treated as a constant for gradient computations, and therefore no gradients are propagated to this term.

Policy error. The policy error was defined as negative of the PPO objective:

$$E_\pi(s, a, \theta) = -\min \left(\frac{\pi_\theta(a|s)}{\bar{\pi}_\theta(a|s)} A^{\pi_\theta}(s, a), \text{clip} \left(\frac{\pi_\theta(a|s)}{\bar{\pi}_\theta(a|s)}, 1 - \epsilon, 1 + \epsilon \right) A^{\pi_\theta}(s, a) \right) - \xi \cdot \pi_\theta(a|s) \log \pi_\theta(a|s). \quad (49)$$

Here, $\xi > 0$ is the entropy coefficient exponentially decaying according to Eq (46), and $[1 - \epsilon, 1 + \epsilon]$ defines the interval of clipping probability ratio $\frac{\pi_\theta(a|s)}{\bar{\pi}_\theta(a|s)}$ as in [82]. Again, the notation $\bar{\pi}$ indicates that this term was treated as a constant during calculation of the error gradient.

The advantage estimate $A^{\pi_\theta}(s, a)$ assuming action a^t is taken at time step t is defined by

$$A^{\pi_\theta}(s^t, a^t) = r_{t+1} + \gamma \bar{V}_\theta(s^{t+1}) - \bar{V}_\theta(s^t), \quad (50)$$

using a temporal difference residual [85] to ensure temporal locality of the learning signal.

For the reinforcement learning tasks, we wanted to enable the model to write and read at every time step. Hence, we let the model perform a memorization event followed by a recall event at each time step t .

Supervised learning. In the supervised learning case, the network output is computed as $y_k^t = \sum_i w_{ki}^{\text{out}} z_i^t$. The supervised error function E_S was defined as cross-entropy between a softmax of the model output vector $\mathbf{y}^t$ during the time step t and target vector $\hat{\mathbf{y}}^t$,

$$E_S^t = \sum_k \hat{y}_k^t \log s_k^t, \quad (51)$$

with $\mathbf{s}^t = \text{softmax}(\mathbf{y}^t)$. Note, that in contrast to the reinforcement learning setup where an error is available each time step t , here the error can only be non-zero at time t where the target signal $\hat{\mathbf{y}}^t$ is received.

Hyperparameters. For all our experiments we used 200 memory neurons, batch size of 32 elements, a global norm gradient clip of 10, and a L2 normalization with a 10^{-4} coefficient.

For our reinforcement learning experiments we used 0.9 as discount rate γ and 0.2 for the PPO clip ratio ϵ .

The task-dependent hyperparameters, as well as the hyperparameters used to obtain the LSTM baseline are reported in Tables 2, 3 and 4. The hyperparameters for both, the memory network model and LSTM were tuned using Hyperband [86] a standard hyperparameter optimization technique, with a similar computational budget. To provide a fair comparison, in addition to hyperparameter tuning, we considered two LSTM configurations, one where the number of LSTM hidden units were the same as our model but with more trainable parameters, and another with the same number of trainable parameters. We report the configuration with the same number of trainable parameters that had the best performance.

Datasets

Omniglot dataset. Omniglot [34] (under MIT License) is a dataset of 1625 handwritten characters from 50 different alphabets, where each character has been drawn by 20 different people. Hence, there are 1625 classes with 20 samples each in this dataset. We pre-trained a 64-dimensional embedding using a 1-shot prototypical network [35] on 965 Omniglot classes. For our experiments, the remaining 660 classes (that were not in the pre-training dataset) were then used, embedded by the previously pre-trained network. Note that these classes were not just held-out samples, but completely new characters.

Table 2. Hyperparameters: Supervised learning.

Hyperparameter	BabiTasks	Association-learning (supervised)
Learning rate λ_{adam}	$5 \cdot 10^{-3}$	$8 \cdot 10^{-3}$
λ_{adam} decay rate	$1 - 10^{-4}$	$1 - 3 \cdot 10^{-4}$
γ_+, γ_-	10^{-1}	10^{-2}
γ_+, γ_- decay rate	$1 - 5 \cdot 10^{-4}$	$1 - 6 \cdot 10^{-3}$

<https://doi.org/10.1371/journal.pone.0313331.t002>

Table 3. Hyperparameters: Reinforcement learning.

Hyperparameter	Association-learning (RL)	Delayed-match-to-sample	Radial-maze
Learning rate λ_{adam}	$8 \cdot 10^{-3}$	$5 \cdot 10^{-3}$	$5 \cdot 10^{-3}$
λ_{adam} decay rate	$1 - 3 \cdot 10^{-4}$	$1 - 3 \cdot 10^{-3}$	$1 - 2 \cdot 10^{-4}$
γ_+, γ_-	10^{-2}	$4.8 \cdot 10^{-1}$	10^{-2}
γ_+, γ_- decay rate	$1 - 6 \cdot 10^{-3}$	$1 - 4 \cdot 10^{-3}$	$1 - 6 \cdot 10^{-4}$
Entropy ξ_0	0.1	0.9	0.9
Entropy decay	$1 - 2 \cdot 10^{-3}$	$1 - 8 \cdot 10^{-4}$	$1 - 4 \cdot 10^{-3}$

<https://doi.org/10.1371/journal.pone.0313331.t003>

Table 4. Hyperparameters: LSTM.

Hyperparameter	Delayed-match-to-sample	Radial-maze
No. of units	55	55
Learning rate λ_{adam}	$5 \cdot 10^{-3}$	$5 \cdot 10^{-3}$
λ_{adam} decay rate	$1 - 3 \cdot 10^{-3}$	$1 - 2 \cdot 10^{-4}$
Entropy ξ_0	0.1	0.69
Entropy decay	$1 - 8 \cdot 10^{-4}$	$1 - 4 \cdot 10^{-3}$

<https://doi.org/10.1371/journal.pone.0313331.t004>

bAbI dataset. The bAbI question answering tasks dataset [17] (under BSD License) is a collection of 20 different tasks, each of which consists of stories written in natural language. A sample can be seen as a sequence of statements (a story), where at the end of each story a question is asked about the content of the previously shown sentences. Depending on the task, the questions often require the combination of multiple statements in order to provide a correct answer. These tasks were designed in a way that they require different skills of natural language processing like induction, deduction, and chaining of facts, which are natural for humans, but hard for machine learning models.

Simulation details for association learning task

The association learning task aimed to test whether the model was capable of learning to memorize and recall association pairs. We performed this task in two different learning scenarios: A reinforcement learning scenario where the model only obtained a reward indicating whether the output action was correct or not, and one supervised learning setup where the cross-entropy error between the network output and the target output was computed.

For this task we compiled two different sets $\mathcal{R} = \{r_1, \dots, r_n\}$ and $\mathcal{S} = \{s_1, \dots, s_n\}$, each consisting of n Omniglot [34] characters from different classes. Each trial in this task consisted of n associations between two stimuli, bijectively mapping each single character from set $\mathcal{R}$ to a single character from set $\mathcal{S}$. Our input vectors for time steps $t \in \{1, \dots, n\}$ were assembled by concatenating two 64-dimensionally embedded vectors $(\mathbf{x}_{\mathcal{R}}^t; \mathbf{x}_{\mathcal{S}}^t)$ (embedding calculated from the prototypical network, see section 'Omniglot dataset' in *Methods* for details), representing one character from set $\mathcal{R}$ and one character from set $\mathcal{S}$ respectively. This resulted in an input time series $\mathbf{x}^1, \dots, \mathbf{x}^n$ according to $(\mathbf{x}_{\mathcal{R}}^1; \mathbf{x}_{\mathcal{S}}^1), \dots, (\mathbf{x}_{\mathcal{R}}^n; \mathbf{x}_{\mathcal{S}}^n)$, followed by a query vector $\mathbf{x}^{n+1} = (\mathbf{x}_{\mathcal{R}}^{n+1}; \mathbf{0})$. In this query the character $\mathbf{x}_{\mathcal{R}}^{n+1}$, which is the same as one of the previously shown $\mathbf{x}_{\mathcal{R}}^\tau$ with $\tau \in \{1, \dots, n\}$, was presented concatenated with a $\mathbf{0}$ -vector instead of its associated $\mathbf{x}_{\mathcal{S}}^\tau$. At each trial, the bijection between $\mathcal{R}$ and $\mathcal{S}$ as well as the sequential order of characters were randomly drawn to ensure an unpredictable set of associations and sequential presentation at each trial.

* *Reinforcement learning variant.* At query time, the model produced a n -dimensional output $\vec{\pi}$ where each element is assigned to a specific character in $\mathcal{S}$. If the model could then successfully predict the previously associated vector $\mathbf{x}_{\mathcal{S}}^\tau$, it obtained a reward of +1. If it failed, it instead received a reward of -1.

For time steps $t \leq n$, where the associative stimuli were presented to the network, no reward was available, since no output action needed to be taken. This principle accounted to a reinforcement learning setup with a delayed reward, where we masked the network outputs during the time steps where the associations were presented, since no meaningful policy error E_π can be computed because no action was required. The value estimate output V^t however was not masked, because it was used to calculate error E_V for the synaptic weight update. The PPO objective was computed as explained in the 'General simulation details' section.

* *Supervised learning variant.* In the supervised case, we provided, as a label, the index of the character in $\mathcal{S}$ bijectively associated with $\mathbf{x}_{\mathcal{R}}^{n+1}$ in the query. The error E_S was then computed for the model prediction at the query presentation. The supervised objective was computed as explained in the 'General simulation details' section.

Simulation details for delayed-match-to-sample task

In the delayed match-to-sample task, a single Omniglot character from a pre-generated set $\mathcal{P}$ of 5 different characters (from different classes) was drawn and presented as 64-dimensional embedded input vector $\mathbf{x}^t$ in the first time step $t = 1$, followed by 8 time steps of white random

noise (with zero mean and standard deviation of 1). The query stimulus was again a character from the same pre-generated set either matching or not matching the previously presented class. We consider 2 different setups (see below for details), where in one setup (the *fixed* case) the input set $\mathcal{P}$ contained a single sample for each class, whereas in the second (the *sampled*) case, $\mathcal{P}$ contained multiple different instances for each class. The model output was a 2-dimensional policy vector π^t representing the *match* or *no match* actions. If the correct action was selected, a reward of +1 was obtained, otherwise -1. As for the association learning task, the policy output π^t for all time steps t where no action was required was masked.

We asked whether the model can generalize to the case where input stimulus and query stimulus have to be matched even if they are not identical, but rather 2 different samples from the same class. Therefore, we evaluated this task in a *fixed* and a *sampled* version. In the *fixed* case, a single instance of each class was added to the input set $\mathcal{P}$. In the case of a match between the two stimuli, the input stimulus and query stimulus were hence always the exact same sample (i.e. the same Omniglot character drawn from the same person). In the *sampled* case, multiple instances of the same class were added to $\mathcal{P}$. Hence, input stimulus and query stimulus could be a different class sample, even though they were from the same class. In this case, the model should also output a *match* decision.

Moreover in the *sampled* case, we used different, non-overlapping input sets for training and testing, further probing generalization capabilities of the model. To accomplish this, we split our set $\mathcal{P}$ into two different subsets, $\mathcal{P}_{\text{train}}$, containing 18 samples for each of the 5 different classes and $\mathcal{P}_{\text{test}}$ containing two sample for the same 5 classes, not contained in $\mathcal{P}_{\text{train}}$. This way we could make sure that the model sees the same classes during training and testing, but not the same instances. The PPO objective was computed as explained in the 'General simulation details' section.

Simulation details for context-dependent reward association (radial maze) task

Task setup. In this task we modelled a radial maze consisting of 8 branches, organized into 4 pairs, with each consecutive branch forming a unique pair, see Fig 5A in the main text. Each pair of branches is associated with a distinct visual context represented by an Omniglot character. During the maze's initialization, only one branch in each pair contains a reward, creating a challenging memorization task for the agent.

Each learning episode consisted of 40 consecutive trials, where, before each episode, the locations of rewarding objects were shuffled. 4 different Omniglot characters were randomly chosen and embedded in 64-dimensional space (using a pre-trained prototypical network, see section 'Omniglot dataset' in *Methods*) to indicate the contexts. Each single trial consisted of two time steps, a query step where the model had to take a decision based on the context and a fact step, where the model received some information about the outcome of the action performed in response to the query.

During a query step at time t for trial s , the input vector $\mathbf{x}^t$ was compiled by concatenating the 64-dimensional context vector $\mathbf{c}^s$ with a 3-dimensional $\mathbf{0}$ -vector. The model computed a 2-dimensional policy vector π^t deciding between the two actions *left* or *right*.

A reward of $\frac{1}{40}$ was given if the rewarding branch was chosen, otherwise the reward was $-\frac{1}{40}$. After obtaining the reward, the model received a new input vector $\mathbf{x}^{t+1} = (\mathbf{c}^s; \mathbf{a}^s; r^s)$ for the fact step. This vector consisted of the concatenation of the context vector $\mathbf{c}^s$, the previously chosen action as 2-dimensional one-hot vector $\mathbf{a}^s$ and a scalar value r^s indicating whether the chosen branch was rewarding ($r^s = 1$) or not ($r^s = 0$). The episode then continued with the query step of the next trial using a new pair with a different context c . More explicitly,

the input time series was $\mathbf{x}^1 = (\mathbf{c}^1; \mathbf{0})$, $\mathbf{x}^2 = (\mathbf{c}^1; \mathbf{a}^1; \mathbf{r}^1)$, $\mathbf{x}^3 = (\mathbf{c}^2; \mathbf{0})$, $\mathbf{x}^4 = (\mathbf{c}^2; \mathbf{a}^2; \mathbf{r}^2)$, ..., $\mathbf{x}^{2S-1} = (\mathbf{c}^S; \mathbf{0})$, $\mathbf{x}^{2S} = (\mathbf{c}^S; \mathbf{a}^S; \mathbf{r}^S)$, with $S = 40$. We tried two different variants of this task: one, where the locations of the rewarding objects for each context c stayed fixed for the whole episode, and one instance where after the fact step, the location of the rewarding object for the previously presented context was switched. The PPO objective was computed as explained in the 'General simulation details' section.

Simulation details for question answering task (bAbI task)

Each task consists of 10,000 data samples, split into training and test set. A sample (or story) consists of a sequence of s sentences called facts, followed by a query sentence q and a target answer a . The j -th word of the i -th sentence in a story can be described as a one-hot encoded vector x_{ij} of size V , where V is the vocabulary size. We consider a sentence i of w words as $f_i = \{x_{i1}, x_{i2}, \dots, x_{iw}\}$. In order to embed each sentence i into a d -dimensional vector, we first generate a word embedding matrix $A \in \mathbb{R}^{d \times V}$ using the He-uniform variance scaling method [41]. We then create a sentence representation that encodes the position of words within a sentence (as proposed in [20]). The authors call this type of representation position encoding (PE). For the sentence f_i , this takes the form: $\mathbf{e}_i = \sum_j \mathbf{l}_j \circ A x_{ij}$, where $\circ$ is the Hadamard product. The column vector $\mathbf{l}_j$ has the structure $l_{kj} = (1 - j/w) - (k/d)(1 - 2j/w)$. The query sentence q is also embedded using A (i.e. $\mathbf{e}_q = \sum_j A q_j$). In our experiment, we applied two types of embedding strategies: first, we generated an embedding as described above and let it fixed for the entire training. We term this method *random embedding* because the word embedding matrix A is randomly sampled for each run. For the second strategy we still generate an embedding as described above, but we trained the embedding end-to-end with our memory model for each task using BPTT, then the embedding was extracted (i.e. word and sentence embedding) and used as pre-trained and fixed embedding for the same model but trained with local synaptic plasticity. We term this method *learned embedding*. We trained our model on each of the 20 bAbI tasks [17] separately at first, then identified which of the tasks were solved (with a test error $< 5\%$) and finally trained the same model again jointly on these tasks. In the joint setup, the stories from the identified tasks were collected together into one training set. Stories were then shuffled before each training epoch. The network was then trained on this training set, without receiving explicit information from which of the tasks each story originated from.

The readout matrix W^{out} was designed in a way that the computed network output vector $\mathbf{y}$ matched vocabulary size V , meaning that the network output (after applying a softmax) was interpreted as a probability vector over words' dictionary. The error was then computed according to Eq (51). In most of the tasks, the output is only a single word, but for example in task 19, the answer could be "n,w" which are technically two words ('n' for north, 'w' for west), concatenated using a comma. To not violate the constraint that we interpret the output of the model as choice for a single word, we defined all n-grams appearing as an answer as a single word and added it to the dictionary. In the "joint" setup, we used a single softmax output layer which represented the vocabulary over all identified tasks. This single output layer was used for all tasks (in the joint setup).

Code source

Source code under GNU General Public License v3.0 is available at <https://github.com/igiTUGraz/tsp>.

Supporting information

S1 Appendix. Additional simulation results.
(PDF)

Author contributions

Conceptualization: Romain Ferrand, Robert Legenstein.

Formal analysis: Romain Ferrand, Maximilian Baronig, Florian Unger.

Funding acquisition: Robert Legenstein.

Investigation: Romain Ferrand.

Methodology: Romain Ferrand.

Project administration: Robert Legenstein.

Resources: Robert Legenstein.

Software: Romain Ferrand.

Supervision: Robert Legenstein.

Visualization: Romain Ferrand.

Writing – original draft: Romain Ferrand, Maximilian Baronig, Robert Legenstein.

Writing – review & editing: Maximilian Baronig, Robert Legenstein.

References

1. Semon RW. The mneme. G. Allen & Unwin Limited; 1921.
2. Tonegawa S, Pignatelli M, Roy DS, Ryan TJ. Memory engram storage and retrieval. *Curr Opin Neurobiol.* 2015;35:101–9. <https://doi.org/10.1016/j.conb.2015.07.009> PMID: 26280931
3. Josselyn SA, Tonegawa S. Memory engrams: recalling the past and imagining the future. *Science.* 2020;367(6473):eaaw4325. <https://doi.org/10.1126/science.aaw4325> PMID: 31896692
4. Mozzachiodi R, Byrne JH. More than synaptic plasticity: role of nonsynaptic plasticity in learning and memory. *Trends Neurosci.* 2010;33(1):17–26. <https://doi.org/10.1016/j.tins.2009.10.001> PMID: 19889466
5. Pignatelli M, Ryan TJ, Roy DS, Lovett C, Smith LM, Muralidhar S, et al. Engram cell excitability state determines the efficacy of memory retrieval. *Neuron.* 2019;101(2):274–284.e5. <https://doi.org/10.1016/j.neuron.2018.11.029> PMID: 30551997
6. Titley HK, Brunel N, Hansel C. Toward a neurocentric view of learning. *Neuron.* 2017;95(1):19–32. <https://doi.org/10.1016/j.neuron.2017.05.021> PMID: 28683265
7. Jacob AD, Mocle AJ, Frankland PW, Josselyn SA. Neuronal excitability in memory allocation: mechanisms and consequences. In: *The oxford handbook of the neurobiology of learning and memory.*
8. Zhang W, Linden DJ. The other side of the engram: experience-driven changes in neuronal intrinsic excitability. *Nat Rev Neurosci.* 2003;4(11):885–900. <https://doi.org/10.1038/nrn1248> PMID: 14595400
9. Gallistel CR, Matzel LD. The neuroscience of learning: beyond the Hebbian synapse. *Annu Rev Psychol.* 2013;64:169–200. <https://doi.org/10.1146/annurev-psych-113011-143807> PMID: 22804775
10. Gallistel CR, Balsam PD. Time to rethink the neural mechanisms of learning and memory. *Neurobiol Learn Mem.* 2014;108:136–44. <https://doi.org/10.1016/j.nlm.2013.11.019> PMID: 24309167
11. Moyer JR Jr, Thompson LT, Disterhoft JF. Trace eyeblink conditioning increases CA1 excitability in a transient and learning-specific manner. *J Neurosci.* 1996;16(17):5536–46. <https://doi.org/10.1523/JNEUROSCI.16-17-05536.1996> PMID: 8757265

12. Moyer JR Jr, Power JM, Thompson LT, Disterhoft JF. Increased excitability of aged rabbit CA1 neurons after trace eyeblink conditioning. *J Neurosci*. 2000;20(14):5476–82. <https://doi.org/10.1523/JNEUROSCI.20-14-05476.2000> PMID: 10884331
13. Janowitz MK, van Rossum MCW. Excitability changes that complement Hebbian learning. *Network*. 2006;17(1):31–41. <https://doi.org/10.1080/09548980500286797> PMID: 16613793
14. Földiák P. Forming sparse representations by local anti-Hebbian learning. *Biol Cybern*. 1990;64(2):165–70. <https://doi.org/10.1007/BF02331346> PMID: 2291903
15. Triesch J. Synergies between intrinsic and synaptic plasticity in individual model neurons. *Advances in neural information processing systems*. 2004. p. 17.
16. Savin C, Joshi P, Triesch J. Independent component analysis in spiking neurons. *PLoS Comput Biol*. 2010;6(4):e1000757. <https://doi.org/10.1371/journal.pcbi.1000757> PMID: 20421937
17. Weston J, Bordes A, Chopra S, Mikolov T. Towards AI-complete question answering: a set of prerequisite toy tasks. In: Bengio Y, LeCun Y, editors. 4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, 2016 May 2–4, Conference Track Proceedings; 2016.
18. Larkum ME, Nevian T, Sandler M, Polsky A, Schiller J. Synaptic integration in tuft dendrites of layer 5 pyramidal neurons: a new unifying principle. *Science*. 2009;325(5941):756–60. <https://doi.org/10.1126/science.1171958> PMID: 19661433
19. Körding KP, König P. Learning with two sites of synaptic integration. *Network*. 2000;11(1):25–39. <https://doi.org/10.1088/0954-898x/11/1/302> PMID: 10735527
20. Sukhbaatar S, Szlam A, Weston J, Fergus R. End-to-end memory networks. In: Cortes C, Lawrence N, Lee D, Sugiyama M, Garnett R, editors. *Advances in Neural Information Processing Systems*. vol. 28. Curran Associates, Inc.; 2015.
21. Limbacher T, Legenstein R. H-Mem: harnessing synaptic plasticity with Hebbian memory networks. *Adv Neural Inf Process Syst*. 2020;33:21627–37.
22. Magee JC, Johnston D. Plasticity of dendritic function. *Curr Opin Neurobiol*. 2005;15(3):334–42. <https://doi.org/10.1016/j.conb.2005.05.013> PMID: 15922583
23. Campanac E, Daoudal G, Ankri N, Debanne D. Downregulation of dendritic I(h) in CA1 pyramidal neurons after LTP. *J Neurosci*. 2008;28(34):8635–43. <https://doi.org/10.1523/JNEUROSCI.1411-08.2008> PMID: 18716222
24. Debanne D, Poo M-M. Spike-timing dependent plasticity beyond synapse - pre- and post-synaptic plasticity of intrinsic neuronal excitability. *Front Synaptic Neurosci*. 2010;2:21. <https://doi.org/10.3389/fnsyn.2010.00021> PMID: 21423507
25. Debanne D, Inglebert Y, Russier M. Plasticity of intrinsic neuronal excitability. *Curr Opin Neurobiol*. 2019;54:73–82. <https://doi.org/10.1016/j.conb.2018.09.001> PMID: 30243042
26. Losonczy A, Makara JK, Magee JC. Compartmentalized dendritic plasticity and input feature storage in neurons. *Nature*. 2008;452(7186):436–41. <https://doi.org/10.1038/nature06725> PMID: 18368112
27. Marti Mengual U, Wybo WAM, Spierenburg LJE, Santello M, Senn W, Nevian T. Efficient low-pass dendro-somatic coupling in the apical dendrite of layer 5 pyramidal neurons in the anterior cingulate cortex. *J Neurosci*. 2020;40(46):8799–815. <https://doi.org/10.1523/JNEUROSCI.3028-19.2020> PMID: 33046549
28. Oja E. A simplified neuron model as a principal component analyzer. *J Math Biol*. 1982;15(3):267–73. <https://doi.org/10.1007/BF00275687> PMID: 7153672
29. Schmidhuber J. Learning to control fast-weight memories: an alternative to dynamic recurrent networks. *Neural Comput*. 1992;4(1):131–9. <https://doi.org/10.1162/neco.1992.4.1.131>
30. Ba J, Hinton GE, Mnih V, Leibo JZ, Ionescu C. Using fast weights to attend to the recent past. *Adv Neural Inf Process Syst*. 2016;29:4331–9.
31. Lillicrap TP, Santoro A. Backpropagation through time and the brain. *Curr Opin Neurobiol*. 2019;55:82–9. <https://doi.org/10.1016/j.conb.2019.01.011> PMID: 30851654
32. Bellec G, Scherr F, Subramoney A, Hajek E, Salaj D, Legenstein R, et al. A solution to the learning dilemma for recurrent networks of spiking neurons. *Nat Commun*. 2020;11(1):3625. <https://doi.org/10.1038/s41467-020-17236-y> PMID: 32681001
33. Gerstner W, Lehmann M, Liakoni V, Corneli D, Brea J. Eligibility traces and plasticity on behavioral time scales: experimental support of NeoHebbian three-factor learning rules. *Front Neural Circuits*. 2018;12:53. <https://doi.org/10.3389/fncir.2018.00053> PMID: 30108488
34. Lake BM, Salakhutdinov R, Tenenbaum JB. Human-level concept learning through probabilistic program induction. *Science*. 2015;350(6266):1332–8. <https://doi.org/10.1126/science.aab3050> PMID: 26659050
35. Snell J, Swersky K, Zemel RS. Prototypical networks for few-shot learning; 2017.

36. Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O. Proximal policy optimization algorithms. arXiv preprint. 2017. <http://arxiv.org/abs/1706.03447>
37. Chudasama Y. In: Stolerman IP, editor. Delayed (Non)match-to-sample task. Berlin, Heidelberg: Springer; 2010. p. 372.
38. van der Maaten L, Hinton G. Visualizing data using t-SNE. *J Mach Learn Res*. 2008;9(86):2579–605.
39. Hochreiter S, Schmidhuber J. Long short-term memory. *Neural Comput*. 1997;9(8):1735–80. <https://doi.org/10.1162/neco.1997.9.8.1735> PMID: 9377276
40. Olton DS, Collison C, Werz MA. Spatial memory and radial arm maze performance of rats. *Learn Motiv*. 1977;8(3):289–314. [https://doi.org/10.1016/0023-9690\(77\)90054-6](https://doi.org/10.1016/0023-9690(77)90054-6)
41. He K, Zhang X, Ren S, Sun J. Delving deep into rectifiers: surpassing human-level performance on imagenet classification. In: Proceedings of the IEEE International Conference on Computer Vision; 2015. p. 1026–34.
42. Lee D, Seung HS. Algorithms for non-negative matrix factorization. *Adv Neural Inf Process Syst*. 2000;13.
43. Quiñero Quiroga R. Neuronal codes for visual perception and memory. *Neuropsychologia*. 2016;83:227–41. <https://doi.org/10.1016/j.neuropsychologia.2015.12.016> PMID: 26707718
44. Ison MJ, Quiñero Quiroga R, Fried I. Rapid encoding of new memories by individual neurons in the human brain. *Neuron*. 2015;87(1):220–30. <https://doi.org/10.1016/j.neuron.2015.06.016> PMID: 26139375
45. Laughlin S. A simple coding procedure enhances a neuron's information capacity. *Z Naturforsch C Biosci*. 1981;36(9–10):910–2. <https://doi.org/10.1515/znc-1981-9-1040> PMID: 7303823
46. Stemmler M, Koch C. How voltage-dependent conductances can adapt to maximize the information encoded by neuronal firing rate. *Nat Neurosci*. 1999;2(6):521–7. <https://doi.org/10.1038/9173> PMID: 10448216
47. Li Y, Li C. Synergies between intrinsic and synaptic plasticity based on information theoretic learning. *PLoS One*. 2013;8(5):e62894. <https://doi.org/10.1371/journal.pone.0062894> PMID: 23671642
48. Schrauwen B, Wardermann M, Verstraeten D, Steil JJ, Stroobandt D. Improving reservoirs using intrinsic plasticity. *Neurocomputing*. 2008;71(7–9):1159–71. <https://doi.org/10.1016/j.neucom.2007.12.020>
49. Steil JJ. Online reservoir adaptation by intrinsic plasticity for backpropagation-decorrelation and echo state learning. *Neural Netw*. 2007;20(3):353–64. <https://doi.org/10.1016/j.neunet.2007.04.011> PMID: 17517491
50. Lazar A, Pipa G, Triesch J. Fading memory and time series prediction in recurrent networks with different forms of plasticity. *Neural Netw*. 2007;20(3):312–22. <https://doi.org/10.1016/j.neunet.2007.04.020> PMID: 17556114
51. Bredenberg C, Lyo B, Simoncelli E, Savin C. Impression learning: online representation learning with synaptic plasticity. *Adv Neural Inf Process Syst*. 2021;34:11717–29.
52. Sandberg A, Tegnér J, Lansner A. A working memory model based on fast Hebbian learning. *Network*. 2003;14(4):789–802. <https://doi.org/10.1088/0954-898x/14/4/309> PMID: 14653503
53. Lansner A, Fiebig F, Herman P. Hebbian fast plasticity and working memory. arXiv preprint. 2023. <http://arxiv.org/abs/2304.06626>
54. Miconi T, Stanley K, Clune J. Differentiable plasticity: training plastic neural networks with backpropagation. In: International Conference on Machine Learning. PMLR; 2018. p. 3559–68.
55. Miconi T, Kay K. An active neural mechanism for relational learning and fast knowledge reassembly. *bioRxiv*. 2023; p. 2023-07.
56. Rohani SRR, Hedayatian S, Baghshah MS. BIMRL: brain inspired meta reinforcement learning. In: 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE; 2022. p. 9048–53.
57. Kumar M, Tan C, Libedinsky C, Yen S, Tan A. One-shot learning of paired associations by a reservoir computing model with Hebbian plasticity. arXiv preprint. 2021. <http://arxiv.org/abs/2106.03580>
58. Yagishita S, Hayashi-Takagi A, Ellis-Davies GCR, Urakubo H, Ishii S, Kasai H. A critical time window for dopamine actions on the structural plasticity of dendritic spines. *Science*. 2014;345(6204):1616–20. <https://doi.org/10.1126/science.1255514> PMID: 25258080
59. He K, Huertas M, Hong SZ, Tie X, Hell JW, Shouval H, et al. Distinct eligibility traces for LTP and LTD in cortical synapses. *Neuron*. 2015;88(3):528–38. <https://doi.org/10.1016/j.neuron.2015.09.037> PMID: 26593091

60. Brzosko Z, Schultz W, Paulsen O. Retroactive modulation of spike timing-dependent plasticity by dopamine. *Elife*. 2015;4:e09685. <https://doi.org/10.7554/eLife.09685> PMID: 26516682
61. Brzosko Z, Zannone S, Schultz W, Clopath C, Paulsen O. Sequential neuromodulation of Hebbian plasticity offers mechanism for effective reward-based navigation. *Elife*. 2017;6:e27756. <https://doi.org/10.7554/eLife.27756> PMID: 28691903
62. Bittner KC, Milstein AD, Grienberger C, Romani S, Magee JC. Behavioral time scale synaptic plasticity underlies CA1 place fields. *Science*. 2017;357(6355):1033–6. <https://doi.org/10.1126/science.aan3846> PMID: 28883072
63. Frey U, Morris RG. Synaptic tagging and long-term potentiation. *Nature*. 1997;385(6616):533–6. <https://doi.org/10.1038/385533a0> PMID: 9020359
64. Brown MA, Sharp PE. Simulation of spatial learning in the Morris water maze by a neural network model of the hippocampal formation and nucleus accumbens. *Hippocampus*. 1995;5(3):171–88. <https://doi.org/10.1002/hipo.450050304> PMID: 7550613
65. Izhikevich EM. Solving the distal reward problem through linkage of STDP and dopamine signaling. *Cereb Cortex*. 2007;17(10):2443–52. <https://doi.org/10.1093/cercor/bhl152> PMID: 17220510
66. Legenstein R, Pecevski D, Maass W. A learning theory for reward-modulated spike-timing-dependent plasticity with application to biofeedback. *PLoS Comput Biol*. 2008;4(10):e1000180. <https://doi.org/10.1371/journal.pcbi.1000180> PMID: 18846203
67. Frémaux N, Sprekeler H, Gerstner W. Reinforcement learning using a continuous time actor-critic framework with spiking neurons. *PLoS Comput Biol*. 2013;9(4):e1003024. <https://doi.org/10.1371/journal.pcbi.1003024> PMID: 23592970
68. Larkum ME, Senn W, Lüscher H-R. Top-down dendritic input increases the gain of layer 5 pyramidal neurons. *Cereb Cortex*. 2004;14(10):1059–70. <https://doi.org/10.1093/cercor/bbh065> PMID: 15115747
69. Larkum M. A cellular mechanism for cortical associations: an organizing principle for the cerebral cortex. *Trends Neurosci*. 2013;36(3):141–51. <https://doi.org/10.1016/j.tins.2012.11.006> PMID: 23273272
70. Manita S, Suzuki T, Homma C, Matsumoto T, Odagawa M, Yamada K, et al. A top-down cortical circuit for accurate sensory perception. *Neuron*. 2015;86(5):1304–16. <https://doi.org/10.1016/j.neuron.2015.05.006> PMID: 26004915
71. Gilbert CD, Li W. Top-down influences on visual processing. *Nat Rev Neurosci*. 2013;14(5):350–63. <https://doi.org/10.1038/nrn3476> PMID: 23595013
72. Kreiman G, Serre T. Beyond the feedforward sweep: feedback computations in the visual cortex. *Ann N Y Acad Sci*. 2020;1464(1):222–41. <https://doi.org/10.1111/nyas.14320> PMID: 32112444
73. Schuman B, Dellal S, Prönnke A, Machold R, Rudy B. Neocortical layer 1: an elegant solution to top-down and bottom-up integration. *Annu Rev Neurosci*. 2021;44:221–52. <https://doi.org/10.1146/annurev-neuro-100520-012117> PMID: 33730511
74. Cai DJ, Aharoni D, Shuman T, Shobe J, Biane J, Song W, et al. A shared neural ensemble links distinct contextual memories encoded close in time. *Nature*. 2016;534(7605):115–8. <https://doi.org/10.1038/nature17955> PMID: 27251287
75. Kastellakis G, Silva AJ, Poirazi P. Linking memories across time via neuronal and dendritic overlaps in model neurons with active dendrites. *Cell Rep*. 2016;17(6):1491–504. <https://doi.org/10.1016/j.celrep.2016.10.015> PMID: 27806290
76. Mozzachiodi R, Byrne JH. More than synaptic plasticity: role of nonsynaptic plasticity in learning and memory. *Trends Neurosci*. 2010;33(1):17–26. <https://doi.org/10.1016/j.tins.2009.10.001> PMID: 19889466
77. Crick F. The recent excitement about neural networks. *Nature*. 1989;337(6203):129–32. <https://doi.org/10.1038/337129a0> PMID: 2911347
78. Lillicrap TP, Cownden D, Tweed DB, Akerman CJ. Random synaptic feedback weights support error backpropagation for deep learning. *Nat Commun*. 2016;7:13276. <https://doi.org/10.1038/ncomms13276> PMID: 27824044
79. Kingma DP, Ba J. Adam: A method for stochastic optimization. *arXiv preprint*. 2014. <http://arxiv.org/abs/1412.6980>
80. Welford BP. Note on a method for calculating corrected sums of squares and products. *Technometrics*. 1962;4(3):419–20. <https://doi.org/10.1080/00401706.1962.10490022>
81. Ioffe S, Szegedy C. Batch normalization: accelerating deep network training by reducing internal covariate shift; 2015.
82. Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O. Proximal policy optimization algorithms. *arXiv preprint*. 2017. <http://arxiv.org/abs/1707.06347>

83. Williams RJ, Peng J. Function optimization using connectionist reinforcement learning algorithms. *Connect Sci.* 1991;3(3):241–68. <https://doi.org/10.1080/09540099108946587>
84. Sutton RS. *Mach Learn.* 1988;3(1):9–44. <https://doi.org/10.1023/a:1022633531479>
85. Schulman J, Moritz P, Levine S, Jordan M, Abbeel P. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint.* 2015. <http://arxiv.org/abs/1506.02438>
86. Li L, Jamieson K, DeSalvo G, Rostamizadeh A, Talwalkar A. Hyperband: a novel bandit-based approach to hyperparameter optimization. *J Mach Learn Res.* 2018;18(185):1–52.