MEMORY-ENRICHED COMPUTATION AND LEARNING IN SPIKING NEURAL NETWORKS THROUGH HEBBIAN PLASTICITY

Thomas Limbacher[a] **Ozan Özdenizci**[a,b] **Robert Legenstein**[a*]

[a]Institute of Theoretical Computer Science, Graz University of Technology, 8010 Graz, Austria
[b]TU Graz - SAL Dependable Embedded Systems Lab, Silicon Austria Labs, 8010 Graz, Austria

{thomas.limbacher, ozan.ozdenizci, robert.legenstein}@igi.tugraz.at

May 24, 2022

ABSTRACT

Memory is a key component of biological neural systems that enables the retention of information over a huge range of temporal scales, ranging from hundreds of milliseconds up to years. While Hebbian plasticity is believed to play a pivotal role in biological memory, it has so far been analyzed mostly in the context of pattern completion and unsupervised learning. Here, we propose that Hebbian plasticity is fundamental for computations in biological neural systems. We introduce a novel spiking neural network architecture that is enriched by Hebbian synaptic plasticity. We show that Hebbian enrichment renders spiking neural networks surprisingly versatile in terms of their computational as well as learning capabilities. It improves their abilities for out-of-distribution generalization, one-shot learning, cross-modal generative association, language processing, and reward-based learning. As spiking neural networks are the basis for energy-efficient neuromorphic hardware, this also suggests that powerful cognitive neuromorphic systems can be build based on this principle.

Keywords Spiking Neural Networks · Hebbian Plasticity · Memory · Few-shot learning

1 Introduction

Spiking Neural Networks (SNNs) are a well-established model of neural computation [1]. In contrast to conventional artificial neural networks (ANNs), neurons in an SNN communicate via stereotypical pulses—so-called spikes—and temporally integrate incoming information in their membrane potential. Since these are key features of biological neurons, SNNs are heavily used to model information processing in the brain. Furthermore, SNNs are well-suited for implementation in neuromorphic hardware, leading to highly energy-efficient AI applications.

For a long time, SNNs have been inferior to ANNs in terms of performance on standard pattern recognition tasks. However, a number of recent advances in SNN research have changed the picture, showing that SNNs can achieve performances similar to ANNs [2]. In particular, the use of surrogate gradients for SNN training [3]–[6] and the use of longer adaptation time constants in recurrent SNNs have been instrumental in this respect [7]. Nevertheless, SNNs still lack many capabilities of their biological counterparts—for some of which biologically implausible ANN solutions have been proposed.

Since computations in SNNs have—in contrast to computations in feed-forward ANNs—a strong temporal component, they have been proposed to be particularly suited for temporal computing tasks [7]. Here, it has turned out that the ability to retain information on several time scales is crucial. The most basic time-constant in spiking neurons is the membrane time constant on the order of tens of milliseconds. In principle, arbitrary time constants can be realized by recurrent connections in recurrent SNNs. However, such recurrent retention of information is rather brittle and hard to learn. Instead, there were several suggestions to utilize longer time constants available in biological neuronal circuits such as short-term plasticity [8], [9] on the order of 100s of milliseconds, and adaptation time constants of neurons

*To whom correspondence should be addressed. E-mail: robert.legenstein@igi.tugraz.at

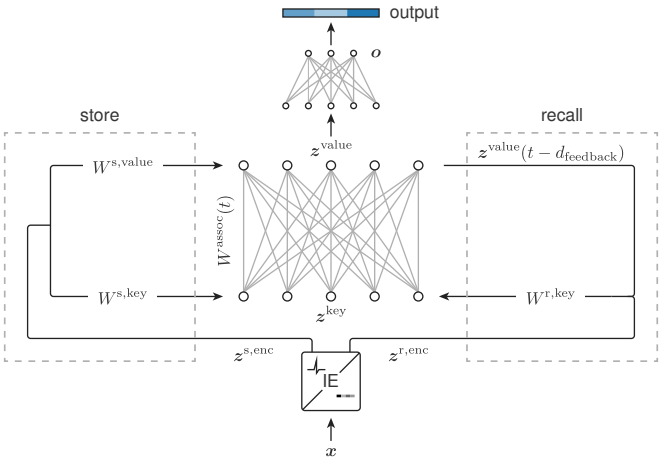

Figure 1: Schematic of the SNN model. Inputs $\boldsymbol{x}$ are encoded by an input encoder (IE). Memory induction: The encoded input $\boldsymbol{z}^{\text{s,enc}}$ activates some neurons in the key- and value-layer through weight matrices $W^{\text{s,key}}$ and $W^{\text{s,value}}$, respectively. Since the key-neurons $\boldsymbol{z}^{\text{key}}$ and the value-neurons $\boldsymbol{z}^{\text{value}}$ are pre- and post-synaptic to the synapses in W^{assoc}, their activity induces weight changes there. Memory recall: The encoded input $\boldsymbol{z}^{\text{r,enc}}$ activates some neurons in the key-layer through the weight matrix $W^{\text{r,key}}$. This activity activates some neurons in the value-layer through synapses W^{assoc}, thus potentially recalling some information that has been stored previously. Finally, value neurons $\boldsymbol{z}^{\text{value}}$ project to a layer of output neurons $\boldsymbol{o}$. To allow for a memory recall based on previously recalled information, activity in the value-layer is fed back to the key-layer with some delay d_{feedback}.

[3], [7] on the order of seconds. The inclusion of such time constants has been shown to extend the computational capabilities of SNNs. However, typical cognitive tasks are frequently situated on a much slower time scale of minutes or longer. For example, when we watch a movie, we have to rapidly memorize facts in order to follow the story and draw conclusions as the narrative evolves. For such tasks, time constants on the order of seconds are insufficient.

Here we consider Hebbian synaptic plasticity [10] as a mechanism to extend the range of time constants and therefore the computational capabilities of SNNs. Hebbian synaptic plasticity is abundant in both neocortex and hippocampus [11]–[13]. While many forms, in particular in sensory cortical areas, are believed to shape processing on a very slow developmental scale, there is also evidence for rapid plasticity that can in principle be utilized for online processing on the behavioral time scale, most prominently in the hippocampus [14], [15]. We show that Hebbian synaptic plasticity renders SNNs rather flexible in terms of their computational as well as learning capabilities. This single principle can give rise to SNNs that are capable of out-of-distribution generalization, one-shot learning, cross-modal generative association, answering questions about stories of various types, and learning to play card games from rewards. Hence, our results show that Hebbian plasticity enhances the computational capabilities of SNNs in several directions. This suggests that Hebbian plasticity is a central component of information processing in the brain which is tightly interwoven with cognitive neuronal processing. Since local Hebbian plasticity can easily be implemented in neuromorphic hardware, this also suggests that powerful cognitive neuromorphic systems can be build based on this principle.

2 Results

2.1 Spiking Neural Networks with associative memory

We consider networks of standard leaky integrate-and-fire (LIF) neurons modeled in discrete time steps Δt. The membrane potential V_j of neuron j at time t is given by

$$V_j(t + \Delta t) = \alpha V_j(t) + (1 - \alpha) I_j(t) - \vartheta z_j(t), \tag{1}$$

where α defines the membrane potential decay per time step. The total synaptic input current $I_j(t) = \sum_i W_{ji} z_i(t)$ is given by the sum over all pre-synaptic neuron's output spike trains weighted by the corresponding synaptic weights W_{ji}. When the neuron's membrane potential $V_j(t)$ is above some threshold ϑ, the neuron spikes ($z_j(t) = 1$) and the membrane potential is reset (last term in (1)). If the neuron does not spike, we define $z_j(t) = 0$.

The considered network model is shown in Fig. 1. At the core of the network is a heteroassociative memory [16], that is, a single layer feed-forward SNN. Here, spiking neurons $\boldsymbol{z}^{\text{key}}$ in the *key-layer* project to neurons $\boldsymbol{z}^{\text{value}}$ in the *value-layer* with synaptic weights W^{assoc}_{ji} that are subject to rapid Hebbian plasticity. We used 100 neurons in the key- and value-layer respectively in all simulations. The use of this simple hetero-associative memory architecture is motivated from the hippocampal circuitry where it was shown that rapid plasticity is found in the connections from region CA3 to CA1, resembling a similar single-layer architecture with key-layer corresponding to CA3 and the

value-layer corresponding to CA1 [14], [15]. From a machine learning perspective, this architecture can be motivated from memory-augmented neural networks, a class of ANN models that were shown to outperform standard ANNs in memory-dependent computational tasks. This class includes networks with key-value memory systems such as memory-networks [17], [18], Hebbian memory networks [19], and transformers [20]. The latter have been shown to be particularly powerful for language-processing, giving rise to language models such as GPT-3 [21].

In our model, neurons in the key-layer (key neurons) receive input from two spiking neuron populations, $\boldsymbol{z}^{\mathrm{s,enc}}$ and $\boldsymbol{z}^{\mathrm{r,enc}}$, responsible for storing (s) and recalling (r) information to and from the memory respectively (see below). Similarly, neurons in the value-layer (value neurons) receive input from two sites, namely from $\boldsymbol{z}^{\mathrm{s,enc}}$ and from the key layer neurons $\boldsymbol{z}^{\mathrm{key}}$. Consider an input $\boldsymbol{x}$ from which some aspects should be stored in memory. It is first encoded by an input encoder (IE in Fig. 1; the architecture of the encoder depends on the task at hand, see below) giving rise to a spike response $\boldsymbol{z}^{\mathrm{s,enc}}$. The synaptic connections to the key-layer activate some key neurons, while the synaptic connections to the value-layer activate some value neurons. As these neurons are pre- and post-synaptic to the synapses in W^{assoc}, this induces weight changes there (see *Methods* for the Hebbian plasticity model). Which information about the input $\boldsymbol{x}$ is stored depends on the synaptic weights from $\boldsymbol{z}^{\mathrm{s,enc}}$ to the key- and value-layers.

Now consider an input $\boldsymbol{x}$ that should trigger a memory recall. In this case, the encoded input $\boldsymbol{z}^{\mathrm{r,enc}}$ activates the key neurons through the weight matrix $W^{\mathrm{r,key}}$ giving rise to activity in the key-layer. This activity activates neurons in the value-layer through association synapses W^{assoc}, thus potentially recalling information that has been stored previously. Finally, these neurons project to a layer of output neurons $\boldsymbol{o}$. For some tasks, it might be beneficial to perform a recall based on previously recalled information. For example, when you are asked "Can Tweety fly?", you may first recall that Tweety is a canary and then remember that canaries can fly to arrive at the correct answer. We therefore included a feedback loop in the model from the value neurons to the key neurons (see loop on the right side of Fig. 1). In this way, recalled activity can influence the recall itself after some delay. We set the feedback delay d_{feedback} to $1\,\mathrm{ms}$ if not otherwise stated.

2.2 Memorizing associations

We first tested the ability of our model to one-shot memorize associations and to use these associations later when needed. Here we conducted experiments on a task that requires to form associations between random continuous-valued vectors and integer labels that were sequentially presented to the network.

In each instance of this task, we randomly drew N real vectors where each element of a vector was sampled from a uniform distribution on the interval $[0, 1)$. We then generated input sequences of N tuples each containing one of the random vectors and an associated label from 1 to N (Fig. 2A, right). After all those vector-label pairs have been presented to the network, it received a query vector. The query vector was equal to one of the N vectors and was randomly selected for each input sequence. The network was required to output the label of the query vector.

The model was trained for 4250 iterations using backpropagation through time (BPTT) [3], [22]. We used two dense layers as input encoder in this task. Each layer consisted of 80 LIF neurons. One layer was used to encode each of the random input vectors and another layer was used to encode the integer labels. Inputs were applied to these layers for $100\,\mathrm{ms}$ each, giving rise to spike trains $\boldsymbol{z}^{\mathrm{s,enc}}$ and $\boldsymbol{z}^{\mathrm{r,enc}}$ (see Section *Model and training details* in the Supplementary for more details to the model and the training setup). Fig. 2A (bottom right) shows the network activity after training for one test example with a sequence length N of eight vector-label pairs.

In Fig. 2B we compare the performance of our model for various sequence lengths (number of vector-label pairs) to the standard generic artificial and spiking recurrent network models: the standard LSTM network [23] and the long short-term memory spiking neural network (LSNN) [3], [7]. The LSTM network consisted of 100 LSTM units. The LSNN consisted of 150 regular spiking and 150 adaptive neurons. For both models, we used the same architecture for the input encoder and the output layer as in our model. For short sequences, the performance of the LSTM network and the LSNN is comparable to our model. While the accuracy of the LSTM and LSNN drastically drops at some point, the accuracy of our model stays above $90\,\%$ for sequences containing up to 50 vector-label pairs.

To test the out-of-distribution generalization capability of our model in this task, we trained it with a sequence length N_{train} of five vector-label pairs and tested the model on shorter and longer sequences of up to 30. Note that the labels in this task were randomly chosen from $\{1, \ldots, 30\}$ and, consequently, the output layer was of size 30. In Fig. 2C, we compare its performance to an LSTM network in terms of the test accuracy. While both models generalize to shorter sequences, our model shows superior generalization to longer sequences.

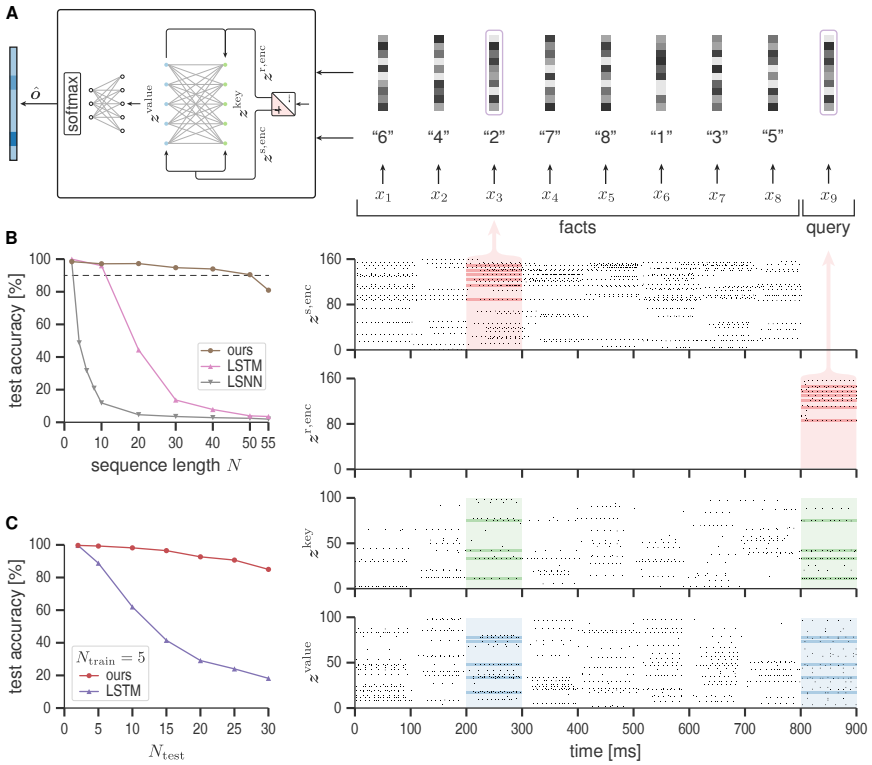

Figure 2: Association task and out-of-distribution generalization. **(A)** Schematic of the network model (left), the network input for an input sequence of length $N = 8$ (top right), and the network activity after training (bottom right). Eight 10-dimensional random vectors along with an associated label from 1 to 8 are presented sequentially as facts (x_1 to x_8). After all those vector-label pairs have been presented to the network, it receives a query vector (x_9). The query vector is equal to one of the vectors presented as facts (randomly selected; vector with label 2 in the shown example). The network is required to output the label of the query vector. Inputs are encoded as spike trains $\boldsymbol{z}^{\text{s,enc}}$ and $\boldsymbol{z}^{\text{r,enc}}$ (100 ms for each input). Synaptic connections to the key-layer activate some key neurons $\boldsymbol{z}^{\text{key}}$, while the synaptic connections to the value-layer activate some value neurons $\boldsymbol{z}^{\text{value}}$. Spikes in these layers during storing of the third vector-label pair x_3 and during recall are shown within red, green, and blue rectangles, respectively. Neurons that are both active during storing of x_3 and recalling are highlighted with saturated color. **(B)** Performance comparison of our model with an LSTM network and an LSNN in this task. Shown is the test accuracy for various sequence lengths N. While the accuracy of our model stays above 90 % for sequences with up to 50 vector-label pairs, the performance of the LSTM network and the LSNN quickly decreases with increasing sequence length. **(C)** Out-of-distribution generalization capability. We trained models with a sequence length of $N_{\text{train}} = 5$ and evaluated the models generalization capability to test sets with shorter and longer sequence lengths N_{test}. Comparison to LSTM network: While both models generalize to new test sets with shorter sequences, our model shows superior generalization to longer sequences.

One-shot Learning

While standard deep learning approaches need large numbers of training examples during training, humans can learn new concepts based on a single exposure (one-shot learning) [24]. A large number of few-shot learning approaches have been proposed using artificial neural networks [25], [26], but biologically plausible SNN models are very rare [27]. We wondered whether Hebbian plasticity could endow SNNs with one-shot learning capabilities. To that end, we applied our model to the problem of one-shot classification on the Omniglot [28] data set. Omniglot consists of 1623 handwritten characters from 50 different alphabets. There are 20 examples of each character, each hand drawn by a different person. Following [29], we resized all the image to 28×28 and augmented the data set with rotations in multiples of 90 degrees. We trained on 1028 characters (4112 classes in total with rotations) and tested on 423 characters (1692 classes with rotations).

In each instance of this task, we randomly drew five different Omniglot classes. We then generated input sequence of tuples each containing a randomly drawn instance of one of the five Omniglot classes and an associated label from 1 to 5 (Fig. 3A right). After all those image-label pairs have been presented to the network, it received a query image. The query image showed another randomly drawn sample from one of these five Omniglot classes. The network was required to output the label that appeared together with an image of the same class as the query image (1-shot 5-way classification).

We trained our model for 200 epochs with 200 iterations per epoch. In each iteration we randomly drew a batch of 256 input sequences, each containing five different Omniglot classes along with an associated label from 1 to 5, and one query image. We used a convolutional neural network (CNN) as input encoder for the Omniglot image, which

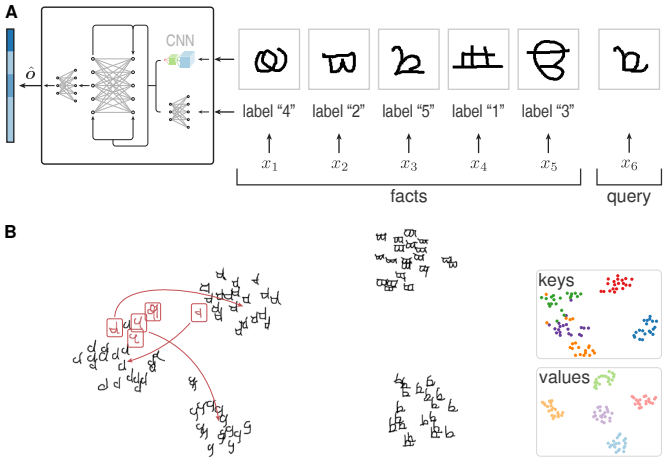

Figure 3: Omniglot one-shot task and a visualization of the embeddings learned by the encoder CNN in this task. **(A)** Schematic of the network model (left) and the network input (right). Five images from the Omniglot data set along with an associated label from 1 to 5 are presented sequentially as facts (x_1 to x_5). After all those image-label pairs have been presented to the network, it receives a query image (x_6). The network is required to output the label that appeared together with an image of the same class as the query image. **(B)** A t-SNE visualization of the embeddings learned by the encoder CNN. A subset of the Tengwar script is shown (an alphabet in the test set). Misclassified characters are highlighted in red along with arrows pointing to the correct cluster. Inset: A t-SNE visualization of the learned key- and value-representation of the inputs. Colors indicate character class.

was pre-trained using the prototypical loss [25] and then converted into a spiking CNN by using a threshold-balancing algorithm [30], [31] (see Section *Converting pre-trained CNNs* in the Supplementary for details to the conversion algorithm). The spiking CNN was then fine-tuned during end-to-end training. We treated the grayscale values of an Omniglot image as a constant input current, applied for $100\,\mathrm{ms}$ to 784 LIF neurons, to produce input spikes to the CNN. The final layer of the CNN consisted of 64 LIF neurons. A single dense layer consisting also of 64 LIF neurons was used to encode the integer labels as spike trains with a duration of $100\,\mathrm{ms}$ per label (see Section *Model and training details* in the Supplementary for more details to the model and the training setup).

The rationale for this network architecture is that, given a suitable generalizing representation of the character, the Hebbian weight matrix can easily associate characters to labels, thus performing one-shot memorization of previously unseen classes to arbitrary labels. In biology, suitable representations could emerge from evolutionary optimized networks potentially fine-tuned by unsupervised plasticity processes. In our setup, these representations are provided by the CNN encoder, where the prototypical loss ensures that similar inputs are mapped to similar representations [25]. The network model can thus be seen as a spiking implementation of a prototypical network. However, the biologically unrealistic nearest-neighbor algorithm used to determine the output of the latter is replaced here by a simple hetero-associative memory. In Fig. 3B we show a sample t-SNE visualization of the embeddings produced by the spiking CNN that was used as image encoder in this task. Despite the shown characters being rather diverse, the network is able to represent them as well-separated clusters. This clustering can also be observed in the learned key- and value-representations of the inputs (inset of Fig. 3B). Overall, the SNN achieved an accuracy of $92.2\,\%$ when tested on 1-shot 5-way classification of novel character classes.

2.3 Cross-modal associations

Humans are able to imagine features of previously encountered stimuli. For example, when you hear the name of a person, you can imagine a mental image of its face. Here, in contrast to the associations considered above, not just classes are associated but (approximate) mental images. We therefore asked whether Hebbian plasticity can enable SNNs to perform such cross-modal associations. We trained our model in an autoencoder-like fashion. We used the FSDD [32] and the MNIST [33] data set in this task. FSDD is an audio/speech data set consisting of recordings of spoken digits. The data set contains 3000 recordings of 6 speakers (50 of each digit per speaker).

In each instance of the task we randomly drew three unique digits between 0 and 9. For each digit we then generated a tuple containing a randomly drawn instance of this digit as audio file from the FSDD data set, and a randomly drawn image of the same digit from the MNIST data set (Fig. 4A right). After these audio-image pairs have been presented to the network, it received an additional audio query. The audio query was another randomly drawn instance from the FSDD data set of one of the previously presented digits. The network was required to generate the image of the handwritten digit that appeared together with the spoken digit of the same class as the audio query.

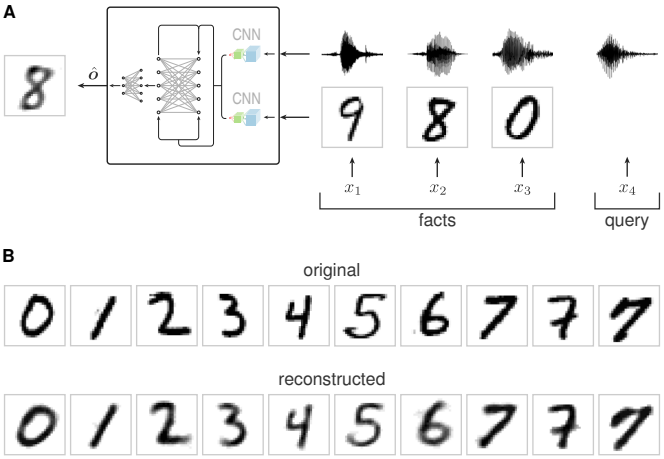

Figure 4: Audio to image synthesis task and examples of generated images. **(A)** Schematic of the network model (left) and the network input (right). Three audio-image pairs are presented sequentially as facts. Input pairs (x_1, x_2, and x_3) contain a spoken digit from the FSDD data set and an image of the same digit from the MNIST data set. After all those audio-image pairs have been presented to the network, it receives an audio query (x_4). The network is required to generate an image of the handwritten digit that appeared together with the spoken digit of the same class as the audio query. **(B)** Example MNIST images from the test set (top) and the corresponding images that were reconstructed by the network (bottom). The reconstructed images are not just typical images for the digit classes, but rather images that are very similar to the images presented previously with the audio cues (compare the three rightmost image pairs).

We used one CNN to encode the audio input (more specifically the mel-frequency cepstrum coefficients (MFCCs), see Section *Model and training details* in the Supplementary), and one CNN to encode the MNIST images. The CNNs were pre-trained on FSDD/MNIST classification tasks respectively and the pre-trained models were converted into spiking CNNs by using the threshold-balancing algorithm [30], [31] (see Section *Converting pre-trained CNNs* in the Supplementary for details to the conversion algorithm). We removed the final classification layers of the CNNs and used the penultimate layer consisting of 64 LIF neurons as the encoding of the input stimuli. The spiking CNN was then fine-tuned during end-to-end training (see Section *Model and training details* in the Supplementary for more details to the model and the training setup). Following the value layer of the model, the image reconstruction was produced by a two-layer fully-connected network with 256 and 784 LIF neurons, respectively.

In Fig. 4B we show example MNIST images from the test set and the corresponding images that were reconstructed by the network. One can see that not just a typical image for the digit class was imagined by the network, but rather an image that is very similar to the image presented previously with the audio cue. This shows that the network did not just memorize the digit class in its associative memory, but rather features that benefit the reconstruction of this specific sample. To quantify the reconstruction performance of our model, we computed the mean squared difference (MSD) between the image produced by the network and all MNIST images in an input sequence. The MSD was 0.03 ± 0.02 (mean $\pm$ standard deviation; median was 0.02 with a lower and upper quartile of 0.01 and 0.03, respectively) between the reconstructed image and the target image, and 0.1 ± 0.04 between the reconstructed image and the two other MNIST images in the input sequence (statistics are over 1000 examples in the test set).

2.4 Question Answering

The bAbI data set [17] is a standard benchmark for cognitive architectures with memory. The data set contains 20 different types of synthetic question-answering (QA) tasks, designed to test a variety of reasoning abilities on stories. Each of these tasks consist of a sequence of sentences followed by a question whose answer is typically a single word (in a few tasks, answers are multiple words; see Table S1 in the Supplementary for example stories and questions). We provided the answer to the model as supervision during training, and it had to predict it at test time on a separate test set. The performance of the model was measured using the average error on the test data set over all tasks and the number of failed tasks (according to the convention of [17], a model had failed to solve a task if the test error was above $5\,\%$ for that task).

Each instance of a task consists of a sequence of M sentences $\langle x_m, \ldots, x_M \rangle$, where the last sentence is a question, and an answer a. We represent each word j in a given sentence x_m by a one-hot vector $\boldsymbol{w}_{m,j}$ of length V (where V is the vocabulary size). We limited the number of sentences in a story to 50 (similar to previous work [18], [34]).

We used a dense layer consisting of 80 LIF neurons as input encoder in this task. We found it helpful to let the model choose for itself which type of sentence encoding to use. We therefore used a learned encoding (see Section *Model and*

Table 1: Test error rates (in %) on the 20 bAbI QA tasks. Comparison of our model to the Spiking RelNet model [35] and to the non-spiking memory-based model H-Mem [19] performing one memory access (1-hop) and three memory accesses (3-hop). Shown are mean error, error on individual tasks, and the number of failed tasks (according to the convention of [17], a model had failed to solve a task if the test error was above 5 % for that task; results of the alternative models were taken from the respective papers). Keys: mem. acc. = number of memory accesses; fb. delay = synaptic delay d_{feedback} in feedback loop.

	Spiking	H-Mem		ours	
	RelNet	mem. acc.		fb. delay	
Task		1-hop	3-hop	1 ms	30 ms
1: Single Supporting Fact	1.0	0.0	0.0	0.0	0.0
2: Two Supporting Facts	-	64.2	0.2	10.4	3.3
3: Three Supporting Facts	-	58.6	26.9	58.6	59.2
4: Two Arg. Relations	0.1	0.0	0.0	0.0	0.0
5: Three Arg. Relations	2.3	4.1	1.3	1.4	2.1
6: Yes/No Questions	0.4	12.2	1.2	28.2	4.2
7: Counting	1.4	0.8	0.8	3.2	3.4
8: Lists/Sets	0.9	0.4	0.5	0.8	0.5
9: Simple Negation	0.7	15.5	3.3	1.4	2.7
10: Indefinite Knowledge	1.7	21.3	1.5	5.8	3.9
11: Basic Coreference	2.1	0.1	0.0	0.0	0.0
12: Conjunction	4.2	0.0	0.0	0.0	0.1
13: Compound Coref.	3.6	2.3	0.0	0.3	0.0
14: Time Reasoning	0.0	7.9	1.1	4.4	2.7
15: Basic Deduction	0.0	1.0	0.0	0.0	0.0
16: Basic Induction	-	54.2	54.8	54.1	53.4
17: Positional Reasoning	2.3	38.8	28.7	37.7	15.7
18: Size Reasoning	0.2	4.8	1.9	0.8	1.8
19: Path Finding	3.7	74.7	77.1	66.5	65.8
20: Agent's Motivations	0.4	0.0	0.0	0.0	0.0
Mean error	1.5	18.0	10.0	13.9	10.9
Failed tasks (err. $> 5\,\%$)	3	9	4	7	4

training details in the Supplementary and [34]). Each sentence was encoded as a spike train with a duration of $100\,\text{ms}$. Similar to previous work [18], [34], we performed three independent runs with different random initializations and report the results of the model with the highest validation accuracy in these runs.

In Table 1 we compare our model to the Spiking RelNet [35] and to the H-Mem model [19], a non-spiking memory network model. Similar to the feedback-loop from the value-layer to the input of the key-layer in our model, the H-Mem model can utilize several memory accesses conditioned on previous memory recalls. The results of H-Mem were reported for a single memory access (1-hop) and three memory accesses (3-hop). It turned out that multiple memory hops were necessary to solve some of the bAbI tasks. Similarly, we found that in our model, an instantaneous feedback loop (with a delay d_{feedback} of $1\,\text{ms}$) struggled with a few tasks that could be solved with a delay of $30\,\text{ms}$, corresponding roughly to three hops during the network inference that lasted $100\,\text{ms}$. We compared the performance of these models in terms of their mean error, error on individual tasks, and the number of failed tasks. Our model with $1\,\text{ms}$ feedback solved 13 of the 20 tasks. By increasing the feedback delay d_{feedback} from $1\,\text{ms}$ to $30\,\text{ms}$ it was able to solve 16 of the 20 tasks. This result indicates that the spiking network can make use of multiple memory accesses in an asynchronous manner, i.e., simply through a delayed feedback loop without the need for discrete memory access steps. The spiking RelNet model solved 17 of the 20. Note however that this model is much more complex, makes heavy use of weight sharing and employs pre-trained LSNNs for word embeddings.

2.5 Reinforcement Learning

While supervisory signals are arguably scarce in nature, it is well-established that animals learn from rewards [36]. In order to test whether memory can also serve SNNs in the reward-based setting, we evaluated our model on an episodic reinforcement learning task. The task is based on the popular children's game *Concentration*. The game *Concentration* requires good memorization skills and is played with a deck of n pairs of cards. The cards in each pair are identical. At the start of each game the cards are shuffled and laid out face down. A player's turn consists of flipping over any two of the cards. If the cards match then they are removed from the game and the current player moves again. Otherwise the cards are turned face down again and the next player proceeds. The player that collects more pairs wins the game.

Here we consider a one-player solitaire version of the game (Fig. 5A). In this version of the game the objective is to find all matching pairs with as few card flips as possible. The cards are arranged on a one-dimensional grid of cells, each of which may be empty or may contain one card. The grid is just large enough to hold all of the $2n$ cards. Each card may either be face up or face down on any given time step (initially all cards are face down). The agent's available actions are to flip over any of the cards at a given time step. More precisely, the action space is an integer from $\{1, 2, \ldots, 2n\}$. Whenever two cards are face up but do not match, they automatically turn face down in the next time step. Whenever

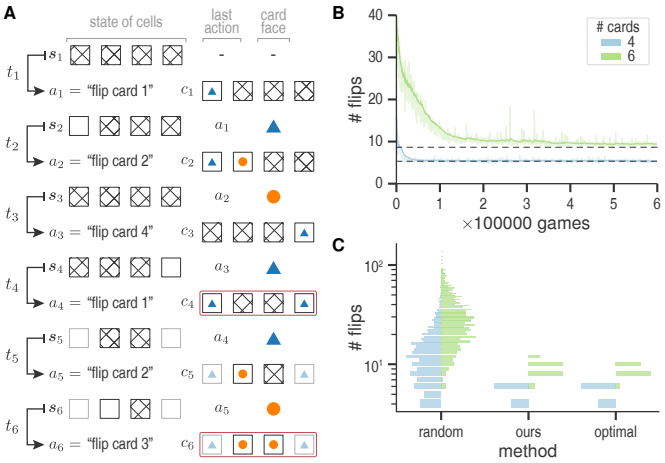

Figure 5: An SNN learns to play the game *Concentration* from rewards. **(A)** Example game moves for a Concentration game with four cards played as a solitaire. The objective of the game is to turn over pairs of matching cards with as few card flips as possible. Shown is—for each time step t_i—the agent's observation $\boldsymbol{s}_i$, the action a_i taken by the agent, and the resulting card configuration c_i after taking action a_i. The agent's observation $\boldsymbol{s}$ contains the state of the cells (face down card, illustrated by square with diamond pattern; face up card, square; or empty, grayed out square), the previous action taken by the agent, and the face of the card the agent had flipped in the previous time step. At time t_1 all cards are face down. Flipping card 1 (action a_1) results in configuration c_1 (i.e., card 1, showing a blue triangle, is face up). Flipping card 2 in time step t_2 reveals an orange disc (configuration c_2). The two cards do not match, hence they are turned face down again in the next time step t_3. In time step t_3, card 4, that shows a blue triangle, is flipped. By recalling that the matching card is card 1, the agent flips card 1 in t_4. Matching face up cards are then removed from the board. The game continues until the remaining two cards are removed. Board configurations at which the agent receives a reward are indicated by a red rectangle. **(B)** Evolution of the number of card flips the agent takes to finish a game over the number of games during training. Shown is the evolution of the number of flips for a deck of four cards (blue) and six cards (green). The mean number of flips is shown as saturated solid line. Black dashed lines show the mean number of flips required when the agent has perfect memory and follows an optimal strategy (5.33 for a deck of four cards and 8.65 for a deck of six cards). **(C)** Histogram of the number of flips an agent takes to finish a game with four cards (blue) and six cards (green). Shown is the histogram for a random agent (left), our agent after training (middle), and an agent that has perfect memory and that follows an optimal strategy (right). Histograms are computed from 1000 games each and are scaled to the same width.

two cards match they are removed from the grid and the agent is rewarded. The agent receives a small penalty for each card flip. The game continues until all cards are removed.

Instead of using images, we define each card face to be a 10-dimensional random continuous-valued vector. The agent's observation vector $\boldsymbol{s}$ (see Fig. 5A) contains three components: (a) a one-hot vector for each cell that encodes the state of the cell (a cell can either be empty, containing a face down card, or a face up card), (b) a one-hot vector encoding the previous action taken by the agent (i.e, a one-hot vector encoding which grid position was flipped), and (c) a 10-dimensional real vector for the image of the card the agent had flipped in the previous time step (this is a zero vector if the card is face down after the flip or if the agent's action was to flip an empty cell). In contrast to the other considered tasks, where we sequentially presented some facts followed by a query to which the network should respond with an output, here instead, in each time step, the network had to figure out by itself when to store and recall information, given only the current observation vector.

The performance was evaluated in terms of the number of flips performed until all matching pairs had been removed from the grid. Agents were trained with proximal policy optimization (PPO) [37] in actor-critic style using 64 asynchronous vectorized environments (see Section *Model and training details* in the Supplementary for details to the model and the training setup). We evaluated our model on a deck of four cards and a deck of six cards. The evolution of the number of card flips the agent takes to finish a game over the number of training steps is shown Fig. 5B. After training we evaluated the agents on 1000 games and recorded the number of card flips the agent takes to finish each game. Fig 5C shows the histogram of the number of card flips in this evaluation for a random agent, our agent, and an agent that has perfect memory and that follows an optimal strategy. If an agent has no memory at all, and plays by simply flipping cards at random, then the expected number of flips the agent takes to finish a game with n pairs of cards is $(2n)^2$. For an optimal agent the length lies between $2n$ and $4n-2$ where the expected number of flips is $(6-4\ln 2)n + 7/8 - 2\ln 2$ as $n \to \infty$ [38]. For the four card-game, the SNN reached an optimal performance (mean number of flips: 5.33; optimal: 5.33). An average of 8 flips were achieved within 8235 games. The six-card game was harder to train. Still, the final network's performance was again close to optimal (see Fig. 5C). Re-drawing the random vectors at the beginning of each game (i.e., using a new deck of cards in each game) marginally decreased the performance (mean number of flips was 5.35 for the four-card game, and 10.88 for the six-card game, where the optimum is 8.65 flips).

3 Discussion

We have presented a novel SNN model that integrates Hebbian plasticity in its network dynamics. We found that this memory-enrichment renders SNNs surprisingly flexible.

While only local plasticity is needed during inference, our model was trained with BPTT. Since BPTT is biologically implausible, we cannot claim that our network is a model for how the functionality could be learned by an organism. Instead, our results provide an existence proof for powerful memory-enhanced SNNs. One can speculate that brain networks were shaped by evolution to make use of Hebbian plasticity processes. In this sense, BPTT can be seen as a replacement for evolutionary processes. For example, the brain might have evolved networks for one-shot learning (Fig. 3) that are particularly tuned to behaviorally relevant stimuli. In addition to evolutionary optimization, local approximations of BPTT such as the recently introduced e-prop algorithm [39] could then further shape the evolved circuits for specific functionality.

The integration of synaptic plasticity for inference in artificial neural networks was used in [40], [41] and adopted recently [42], [43]. In the latter, input representations were bound to labels with Hebbian plasticity. Memory-augmented neural networks use explicit memory modules which are a differentiable version of a digital memory [17], [18], [44]–[48]. Our model utilizes biological Hebbian plasticity instead. In [49], training of networks with synaptic plasticity was explored, but there the parameters of the plasticity rule were optimized instead of the surrounding control networks.

The inclusion of Hebbian plasticity can be viewed as the introduction of another long time constant in the network dynamics. Previous work has shown that longer time constants can significantly improve the temporal computing capabilities of SNNs. In this direction, short-term synaptic plasticity [8], [9] and neuronal adaptation [7] have been exploited. One-shot learning of SNNs has been studied in [27]. Instead of Hebbian plasticity, this model relied on a more elaborate three-factor learning rule. Another SNN model, the Spiking RelNet, was tested on the bAbI task set [35]. We have compared this model to ours in Table 1. The architecture of this model is quite different from our proposal. As a spiking implementation of relational networks, it is rather complex and makes heavy use of weight sharing. Both these models perform well on similar tasks of the bAbI task set, but interestingly, there are some differences. For example, our model solves task 2 "two supporting facts" on which the Spiking RelNet fails. This might be due to the possibility of multiple memory accesses through the feedback loop in our model. On the other hand, our model fails at task 17 "positional reasoning" which is solved by the Spiking RelNet. We suspect that this is due to the more complex network structure of the Spiking RelNet. To the best of our knowledge, no previous spiking (or artificial) neural network model is performing well on both, one-shot learning and bAbI tasks, as well as on the other tasks we presented.

Hebbian plasticity is spatially and temporally local, i.e., the synaptic weight change depends only on a filtered version of the pre- and post-synaptic spikes and on the current weight value. This is a very desirable feature of any plasticity rule, as it can easily be implemented both in biological synaptic connections, and in neuromorphic hardware. In fact, current neuromorphic designs support this type of plasticity [50]. Hence, our results indicate that Hebbian plasticity can serve as a fundamental building block in cognitive architectures based on energy-efficient neuromorphic hardware.

4 Methods

4.1 Neuron Model

Neurons in our model are standard LIF neurons modeled in discrete time steps Δt. The membrane potential V_j of a neuron j at time t is given by

$$V_j(t+\Delta t) = \alpha V_j(t) + (1-\alpha) I_j(t) - \vartheta z_j(t), \tag{2}$$

where $\alpha = \exp(-\frac{\Delta t}{\tau_{\mathrm{m}}})$ defines the membrane potential decay per time step. The total synaptic input current I_j is defined as the weighted sum of spikes from pre-synaptic neurons

$$I_j(t) = \sum_i W_{ji} z_i(t), \tag{3}$$

where the sum goes over all pre-synaptic neurons i and where W_{ji} is the weight of the corresponding synapse. A neuron j spikes as soon as its membrane potential V_j exceeds a firing threshold ϑ. The spike train of a neuron j is modeled as binary sequence $z_j \in \{0,1\}$. After having fired a spike ($z_j(t)=1$), the neurons membrane potential is reset by subtracting the threshold value ϑ (last term in (2)) and the neuron enters an absolute refractory period Δ_{abs} where it cannot spike again (for parameter values, see Table 2).

On the basis of the above formalism, it is apparent that only the spikes are communicated to other neurons. Hence, we interpret z_j as the output of a neuron j and V_j as its hidden state. For simplicity, in the following, we will not state the hidden dynamics of the neurons in the model, but only the synaptic input current and the output spike train.

4.2 Model

Our model takes a sequence of inputs $\langle x_1, \ldots, x_M \rangle$. Each input x_m can either be a fact or a query to which the network should respond with an output $\hat{\boldsymbol{o}}$.

4.2.1 Input encoder

Let $\langle x_1, \ldots, x_M \rangle$ be the given input sequence. Each x_m is converted into a spike train of duration $\tau_{\text{sim}} = 100\,\text{ms}$ by using an input encoder (IE). We do not restrict the type of input encoder. It has to be chosen for the task at hand. In the simplest case we use a single dense layer $\mathcal{E}$ consisting of d LIF neurons. Depending on whether the input x_m represents some fact that might be useful to store, or a query to which the network should respond with an output, we denote the resulting spike train as $z_i^{\text{s,enc}}$ and $z_i^{\text{r,enc}}$, respectively.

4.2.2 Memory induction

Consider an input from which some aspect should be stored in memory. Here, the spike train encoding that input, that is $z_i^{\text{s,enc}}$, is fed to two single-layer networks $\mathcal{K}$ and $\mathcal{V}$ comprising of l LIF neurons each. The input current to a neuron j in layer $\mathcal{K}$—we call it the key-layer—is given by

$$I_j^{\text{key}}(t) = \sum_i W_{ji}^{\text{s,key}} z_i^{\text{s,enc}}(t), \tag{4}$$

where $W^{\text{s,key}}$ is a synaptic-weight matrix of size $l \times d$. We denote the spike train of a neuron j in the key-layer as z_j^{key}. The input current to a neuron k in the value-layer, layer $\mathcal{V}$, is given by

$$I_k^{\text{value}}(t) = \sum_i W_{ki}^{\text{s,value}} z_i^{\text{s,enc}}(t) + c \sum_j W_{kj}^{\text{assoc}}(t) z_j^{\text{key}}(t), \tag{5}$$

where $W^{\text{s,value}}$ is a synaptic-weight matrix of size $l \times d$, $c = 0.2$ is a constant, and where W^{assoc} are association synapses represented as a square matrix of size $l \times l$. The first term represents the contribution of the spikes from the input encoder to this current, and the second term represent the contribution of the spikes that travel from the key-layer via W^{assoc} to a neuron in layer $\mathcal{V}$. We denote the spike train of a neuron k in this layer as z_k^{value}.

Synapses W^{assoc} are subject to Hebbian plasticity. An association between the activity in the key- and value-layer neurons is established via weight changes given by:

$$\Delta W_{kj}^{\text{assoc}}(t) = \gamma_+ (w^{\max} - W_{kj}^{\text{assoc}}(t))\, \kappa_k^{\text{value}}(t) \kappa_j^{\text{key}}(t) - \gamma_- W_{kj}^{\text{assoc}}(t) \kappa_j^{\text{key}}(t)^2, \tag{6}$$

where $\gamma_+ > 0$, $\gamma_- > 0$, $w^{\max}$ are constants, and where κ_j^{key} and κ_k^{value} are exponential activity traces of z_j^{key} and z_k^{value}, respectively (for parameter values, see Table 2). Traces are updated by an amount $1 - \exp(-\Delta t / \tau_{\text{trace}})$ at the moment of spike arrival and decay exponentially with time constant τ_{trace} in the absence of spikes.

The first term in (6) implements a soft upper bound $w^{\max}$ on the weights. The Hebbian term $\kappa_k^{\text{value}}(t)\kappa_j^{\text{key}}(t)$ strengthens connections between co-active neurons in the key- and value-layers. Finally, the last term generally weakens connections from the currently active key-neurons. Since the Hebbian component strengthens connections to active value-neurons, this emphasizes the current association and de-emphasizes old ones. This update is similar to Oja's rule [51], but note that the quadratic term acts on the pre-synaptic neuron. Association synapses are then updated according to $W^{\text{assoc}}(t + \Delta t) = W^{\text{assoc}}(t) + \Delta W^{\text{assoc}}(t)$.

4.2.3 Memory recall

Now consider an input that should trigger a memory recall. In this case, we feed the encoded input $z_i^{\text{r,enc}}$ and, via delayed feedback connections, the activity in the value layer z_k^{value} to the key-layer $\mathcal{K}$. The input current to a neuron j in this layer is then given by

$$I_j^{\text{key}}(t) = \sum_{i \le d} W_{ji}^{\text{r,key}} z_i^{\text{r,enc}}(t) + \sum_{d \le k \le d+l} W_{jk}^{\text{r,key}} z_{k-d}^{\text{value}}(t - d_{\text{feedback}}), \tag{7}$$

where $W^{\text{r,key}}$ is a synaptic-weight matrix of size $l \times d + l$, and where d_{feedback} is the synaptic delay in the feedback connections. As before, we denote the spike train of a neuron j in this layer as z_j^{key}. This activity activates some

Table 2: Neuron and plasticity parameters.

ϑ	Firing threshold	0.1
Δ_{abs}	Refractory period	3 ms
τ_{m}	Membrane time constant	20 ms
τ_{trace}	Time constant of synaptic trace	20 ms
$w^{\max}$	Soft maximum of Hebbian weights	1.0
γ_+	Write factor of Hebbian update rule	0.3
γ_-	Forget factor of Hebbian update rule	0.3

neurons in the value-layer through synapses W^{assoc}, thus potentially recalling some information that has been stores previously. The synaptic current to a neuron k in the value-layer is given by

$$I_k^{\text{value}}(t) = \sum_j W_{kj}^{\text{assoc}}(t) z_j^{\text{key}}(t). \tag{8}$$

As before, we denote the spike train of a neuron k in this layer as z_k^{value}.

4.2.4 Generating the final prediction

The architecture of the output module depends on the task at hand. In the simplest case, the network output was determined by taking the sum of z_k^{value} over the last τ_{read} time steps and passing the result through a final weight matrix W_{out}:

$$\hat{o}_j = \sum_k W_{jk}^{\text{out}} \sum_{t'=0}^{\tau_{\text{read}}} z_k^{\text{value}}(M\tau_{\text{sim}} - t'). \tag{9}$$

The weights $W^{\text{s,key}}$, $W^{\text{s,value}}$, $W^{\text{r,key}}$, and W^{out} are learned during training by minimizing the cross-entropy loss between the softmax of $\hat{\boldsymbol{o}}$ and the target output $\boldsymbol{o}$ using the Adam optimizer [52]. The gradient is backpropagated through spikes by replacing the non-existent derivative of the membrane potential at the time of a spike by a pseudo-derivative, as done in [3], that smoothly increases from 0 to 1, and then decays back to 0:

$$\frac{dz_j(t)}{dv_j(t)} \coloneqq \beta \max\{0, 1 - |v_j(t)|\} \tag{10}$$

where $v_j(t) = \frac{V_j(t) - \vartheta}{\vartheta}$ is the normalized membrane potential of neuron j and $\beta \leq 1$ is a dampening factor which was set to 1. Association synapses W^{assoc} were represented by a square matrix which was initialized for each input sequence $\langle x_m \rangle$ with all its values set to zeros.

Data Availability

The Omniglot data set [28] is freely available at https://github.com/brendenlake/omniglot. The FSDD data set [32] is freely available at https://github.com/Jakobovski/free-spoken-digit-dataset. The MNIST data set [33] is freely available at http://yann.lecun.com/exdb/mnist. The bAbI data set [17] is freely available at https://research.fb.com/downloads/babi.

Code Availability

The models were implemented in PyTorch [53] and the code is available at https://github.com/IGITUGraz/MemoryDependentComputation.

ACKNOWLEDGMENTS

This work was supported by the CHIST-ERA grant CHIST-ERA-18-ACAI-004, by the Austrian Science Fund (FWF) project number I 4670-N (project SMALL), and by the "University SAL Labs" initiative of Silicon Austria Labs (SAL). We thank Wolfgang Maass and Arjun Rao for initial discussions.

References

[1] W. Maass and C. M. Bishop, *Pulsed neural networks*. MIT press, 2001.

[2] A. Tavanaei, M. Ghodrati, S. R. Kheradpisheh, T. Masquelier, and A. Maida, "Deep learning in spiking neural networks," *Neural networks*, vol. 111, pp. 47–63, 2019.

[3] G. Bellec, D. Salaj, A. Subramoney, R. Legenstein, and W. Maass, "Long short-term memory and learning-to-learn in networks of spiking neurons," in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31, Curran Associates, Inc., 2018. [Online]. Available: https://proceedings.neurips.cc/paper/2018/file/c203d8a151612acf12457e4d67635a95-Paper.pdf.

[4] D. Huh and T. J. Sejnowski, "Gradient descent for spiking neural networks," *Advances in neural information processing systems*, vol. 31, 2018.

[5] F. Zenke and S. Ganguli, "Superspike: Supervised learning in multilayer spiking neural networks," *Neural computation*, vol. 30, no. 6, pp. 1514–1541, 2018.

[6] E. O. Neftci, H. Mostafa, and F. Zenke, "Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks," *IEEE Signal Processing Magazine*, vol. 36, no. 6, pp. 51–63, 2019.

[7] D. Salaj, A. Subramoney, C. Kraisnikovic, G. Bellec, R. Legenstein, and W. Maass, "Spike frequency adaptation supports network computations on temporally dispersed information," *Elife*, vol. 10, e65459, 2021.

[8] W. Maass, T. Natschläger, and H. Markram, "Real-time computing without stable states: A new framework for neural computation based on perturbations," *Neural computation*, vol. 14, no. 11, pp. 2531–2560, 2002.

[9] G. Mongillo, O. Barak, and M. Tsodyks, "Synaptic theory of working memory," *Science*, vol. 319, no. 5869, pp. 1543–1546, 2008.

[10] D. O. Hebb, *The organization of behavior: A neuropsychological theory*. Psychology Press, 2005.

[11] T. V. Bliss and T. Lømo, "Long-lasting potentiation of synaptic transmission in the dentate area of the anaesthetized rabbit following stimulation of the perforant path," *The Journal of physiology*, vol. 232, no. 2, pp. 331–356, 1973.

[12] R. C. Malenka and R. A. Nicoll, "Long-term potentiation–a decade of progress?" *Science*, vol. 285, no. 5435, pp. 1870–1874, 1999.

[13] M.-S. Rioult-Pedotti, D. Friedman, and J. P. Donoghue, "Learning-induced ltp in neocortex," *science*, vol. 290, no. 5491, pp. 533–536, 2000.

[14] K. C. Bittner, A. D. Milstein, C. Grienberger, S. Romani, and J. C. Magee, "Behavioral time scale synaptic plasticity underlies ca1 place fields," *Science*, vol. 357, no. 6355, pp. 1033–1036, 2017.

[15] X. Zhao, C.-L. Hsu, and N. Spruston, "Rapid synaptic plasticity contributes to a learned conjunctive code of position and choice-related information in the hippocampus," *Neuron*, vol. 110, no. 1, pp. 96–108, 2022.

[16] D. J. Willshaw, O. P. Buneman, and H. C. Longuet-Higgins, "Non-holographic associative memory," *Nature*, vol. 222, no. 5197, pp. 960–962, 1969.

[17] J. Weston, A. Bordes, S. Chopra, A. M. Rush, B. Van Merriënboer, A. Joulin, and T. Mikolov, "Towards AI-complete question answering: A set of prerequisite toy tasks," arXiv preprint arXiv:1502.05698, 2015.

[18] S. Sukhbaatar, A. Szlam, J. Weston, and R. Fergus, "End-to-end memory networks," in *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., vol. 28, Curran Associates, Inc., 2015. [Online]. Available: https://proceedings.neurips.cc/paper/2015/file/8fb21ee7a2207526da55a679f0332de2-Paper.pdf.

[19] T. Limbacher and R. Legenstein, "H-mem: Harnessing synaptic plasticity with hebbian memory networks," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds., vol. 33, Curran Associates, Inc., 2020, pp. 21 627–21 637. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/f6876a9f998f6472cc26708e27444456-Paper.pdf.

[20] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30, Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[21] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33, Curran Associates, Inc., 2020, pp. 1877–1901. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

[22] P. J. Werbos, "Backpropagation through time: What it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[23] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[24] B. M. Lake, T. D. Ullman, J. B. Tenenbaum, and S. J. Gershman, "Building machines that learn and think like people," *Behavioral and brain sciences*, vol. 40, 2017.

[25] J. Snell, K. Swersky, and R. Zemel, "Prototypical networks for few-shot learning," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30, Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/cb8da6767461f2812ae4290eac7cbc42-Paper.pdf.

[26] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proceedings of the 34th International Conference on Machine Learning*, D. Precup and Y. W. Teh, Eds., ser. Proceedings of Machine Learning Research, vol. 70, PMLR, 2017, pp. 1126–1135. [Online]. Available: https://proceedings.mlr.press/v70/finn17a.html.

[27] F. Scherr, C. Stöckl, and W. Maass, "One-shot learning with spiking neural networks," bioRxiv preprint bioRxiv:2020.06.17.156513, 2020.

[28] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum, "Human-level concept learning through probabilistic program induction," *Science*, vol. 350, no. 6266, pp. 1332–1338, 2015.

[29] O. Vinyals, C. Blundell, T. Lillicrap, k. kavukcuoglu koray, and D. Wierstra, "Matching networks for one shot learning," in *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29, Curran Associates, Inc., 2016, pp. 3630–3638. [Online]. Available: https://proceedings.neurips.cc/paper/2016/file/90e1357833654983612fb05e3ec9148c-Paper.pdf.

[30] P. U. Diehl, D. Neil, J. Binas, M. Cook, S.-C. Liu, and M. Pfeiffer, "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," in *International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2015, pp. 1–8.

[31] A. Sengupta, Y. Ye, R. Wang, C. Liu, and K. Roy, "Going deeper in spiking neural networks: VGG and residual architectures," *Frontiers in Neuroscience*, vol. 13, p. 95, 2019.

[32] Z. Jackson, *Spoken digit*, https://github.com/Jakobovski/free-spoken-digit-dataset, 2016.

[33] Y. LeCun, C. Cortes, and C. Burges, "MNIST handwritten digit database," *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, vol. 2, 2010.

[34] M. Henaff, J. Weston, A. Szlam, A. Bordes, and Y. LeCun, "Tracking the world state with recurrent entity networks," arXiv preprint arXiv:1612.03969, 2017.

[35] P. Plank, A. Rao, A. Wild, and W. Maass, "A long short-term memory for AI applications in spike-based neuromorphic hardware," arXiv preprint arXiv:2107.03992, 2021.

[36] W. Schultz, "Getting formal with dopamine and reward," *Neuron*, vol. 36, no. 2, pp. 241–263, 2002.

[37] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," arXiv preprint arXiv:1707.06347, 2017.

[38] D. J. Velleman and G. S. Warrington, "What to expect in a game of memory," *The American Mathematical Monthly*, vol. 120, no. 9, pp. 787–805, 2013.

[39] G. Bellec, F. Scherr, A. Subramoney, E. Hajek, D. Salaj, R. Legenstein, and W. Maass, "A solution to the learning dilemma for recurrent networks of spiking neurons," *Nature communications*, vol. 11, no. 1, pp. 1–15, 2020.

[40] G. E. Hinton and D. C. Plaut, "Using fast weights to deblur old memories," in *Proceedings of the ninth annual conference of the Cognitive Science Society*, 1987, pp. 177–186.

[41] J. Schmidhuber, "Learning to control fast-weight memories: An alternative to dynamic recurrent networks," *Neural Computation*, vol. 4, no. 1, pp. 131–139, 1992.

[42] J. Ba, G. E. Hinton, V. Mnih, J. Z. Leibo, and C. Ionescu, "Using fast weights to attend to the recent past," in *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29, Curran Associates, Inc., 2016, pp. 4331–4339. [Online]. Available: https://proceedings.neurips.cc/paper/2016/file/9f44e956e3a2b7b5598c625fcc802c36-Paper.pdf.

[43] T. Munkhdalai and A. Trischler, "Metalearning with hebbian fast weights," arXiv preprint arXiv:1807.05076, 2018.

[44] A. Graves, G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-Barwińska, S. G. Colmenarejo, E. Grefenstette, T. Ramalho, J. Agapiou, *et al.*, "Hybrid computing using a neural network with dynamic external memory," *Nature*, vol. 538, no. 7626, pp. 471–476, 2016.

[45] A. Miller, A. Fisch, J. Dodge, A.-H. Karimi, A. Bordes, and J. Weston, "Key-value memory networks for directly reading documents," arXiv preprint arXiv:1606.03126, 2016.

[46] C. Xiong, S. Merity, and R. Socher, "Dynamic memory networks for visual and textual question answering," in *International conference on machine learning*, 2016, pp. 2397–2406.

[47] S. Chandar, S. Ahn, H. Larochelle, P. Vincent, G. Tesauro, and Y. Bengio, "Hierarchical memory networks," arXiv preprint arXiv:1605.07427, 2016.

[48] C. Gulcehre, S. Chandar, K. Cho, and Y. Bengio, "Dynamic neural turing machine with continuous and discrete addressing schemes," *Neural computation*, vol. 30, no. 4, pp. 857–884, 2018.

[49] T. Miconi, K. Stanley, and J. Clune, "Differentiable plasticity: Training plastic neural networks with backpropagation," in *International Conference on Machine Learning*, 2018, pp. 3559–3568.

[50] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[51] E. Oja, "Simplified neuron model as a principal component analyzer," *Journal of mathematical biology*, vol. 15, no. 3, pp. 267–273, 1982.

[52] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," arXiv preprint arXiv:1412.6980, 2014.

[53] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[54] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, JMLR Workshop and Conference Proceedings, 2010, pp. 249–256.

[55] A. M. Saxe, J. L. Mcclelland, and S. Ganguli, "Exact solutions to the nonlinear dynamics of learning in deep linear neural network," in *In International Conference on Learning Representations*, 2014.

Supplementary

Model and training details

Here we give details to our models, and to the encoding and representation of inputs used in our models. The models were simulated with a time step Δt of $1\,\mathrm{ms}$. We used the Euler method for the integration of the neuron dynamics. Every neuron state, that is, membrane potential, remaining refractory period, etc. are represented by fixed size state vectors that are updated at each time step of the simulation.

Spike rate regularization

To achieve biologically plausible neuron and network activity, we used an L_2 activity regularizer, which encourages solutions with sparse spiking. For any particular layer with N neurons, the spike rate regularization loss is calculated according to

$$L_\rho = \lambda_\rho \frac{1}{N} \sum_{i=1}^{N} (\rho_0 - \rho_i)^2, \tag{1}$$

where λ_ρ is the regularization factor, $\rho_0 = 0$ is the target firing rate, and ρ_i is the average firing rate of neuron i. The average is across the entire simulation duration of a given layer and the entire batch.

Converting pre-trained CNNs

We considered the pre-trained CNN parameters as the synaptic weights while computing the weighted sum-of-spikes from pre-synaptic neurons via convolution operations. We replaced the post-convolution ReLU activations with integrate-and-fire (IF) neuron dynamics which implements $V_j(t + \Delta t) = V_j(t) + I_j(t) - \vartheta z_j(t)$ without membrane potential leaks within the CNN layers, as opposed to the LIF neurons which are used in our models otherwise. Finally, we determine the layer-wise IF neuron firing thresholds within the CNN using the ANN-to-SNN conversion by threshold-balancing algorithm [30], [31]. The algorithm obtains a firing threshold for each CNN layer sequentially. We initialize all four layer-wise CNN firing thresholds with zeros. We treat the model inputs (i.e., grayscale Omniglot/MNIST pixel intensities or MFCC values) as a constant current applied for $100\,\mathrm{ms}$ to LIF neurons, to produce input spikes to this CNN. We used training set samples converted into input spikes (all training samples for MNIST and MFCC encoders, and $12\,800$ randomly drawn images for Omniglot which we found to be sufficient to obtain stable threshold estimates) and performed an initial forward pass from the first layer of the CNN. While doing so, we record the value of the maximum input current (incoming weighted sum-of-spikes) observed across all used training samples at any time step during the $100\,\mathrm{ms}$, and set the firing threshold of the first CNN layer to have this value at the end. After the firing threshold of the first layer is set, we perform a forward pass of these spiking inputs starting from the first CNN layer once again using the set firing threshold, and then determine the firing threshold for the second layer in a similar way. We continue this process sequentially for all four layers of the CNN to determine layer-wise firing thresholds.

Following ANN-to-SNN conversion, spiking CNN synaptic weights were fine-tuned during end-to-end training of the models. At this stage, in cross-modal associations experiments, we also introduced additional (non-zero) bias terms for all convolution kernels in the CNNs. Note that this allows us to also fine-tune the conversion-based preset layer-wise firing thresholds individually for each CNN convolution kernel. In one-shot learning experiments, however, we did not (find it beneficial to) introduce bias terms to fine-tune the preset layer-wise CNN firing thresholds.

Details to: Memorizing associations

We used two dense layers as input encoder in this task. Each layer consisted of 80 LIF neurons. One layer was used to encode each of the random input vectors and another layer was used to encode the integer labels. We trained models using different values for N (ranging from 2 to 55). We tuned the hyper-parameters using a validation set which contained 1000 sequences. The models were tested on 2000 sequences. We generated a data set for every iteration of BPTT. We trained for 4250 iterations using mini batches of size 512. The models were trained with Adam [52] using a learning rate of $\mu = 0.003$, that was reduced by $15\,\%$ every 340 iterations. The output of the network was produced by passing the number of spikes over the last $30\,\mathrm{ms}$ of $\boldsymbol{z}^{\mathrm{value}}$ through a final output layer with weights W^{out} and a softmax. The error was computed when the network produced its prediction after the query and gradients were propagated through all time steps of the computation. Weights were initialized using Glorot uniform initialization [54] with a gain of $\sqrt{2}$. The feedback delay d_{feedback} was set to $1\,\mathrm{ms}$. Starting from iteration 0, we applied L_2 activity regularization to all spiking neurons of the model (see Section *Spike rate regularization* for details). The regularization factor λ_ρ was set to 1×10^{-5}. Gradients with an L_2-norm larger than 40.0 were normalized to have norm 40.0.

Details to: One-shot Learning

We used a CNN as input encoder for the Omniglot images. The CNN consisted of four weight layers and had the following structure: $4 \times$ (Conv2D $\rightarrow$ Activation $\rightarrow$ MaxPooling) $\rightarrow$ Flatten. Each block comprises a 64-filter convolution with a kernel size of 3×3 (stride = 1, zero-padding = 1 for each dimension) without bias terms. We used a 2×2 pool size in the max pooling layers. When applied to the 28×28 Omniglot images this architecture results in a 64-dimensional output space. This CNN was pre-trained using the prototypical loss [25] with a ReLU function as the activation after convolutions (i.e., as a conventional ANN). We converted the pre-trained CNN into a spiking CNN that uses IF neuron dynamics as post-convolution activations, and used a threshold-balancing algorithm to set layer-wise firing thresholds [30], [31] (detailed in Section *Converting pre-trained CNNs*). The converted spiking CNN synaptic weights were later fine-tuned during end-to-end training. We treated the grayscale values of an Omniglot image as a constant input current, applied for $100\,\mathrm{ms}$ to 784 LIF neurons, to produce input spikes to this CNN. The final layer of the CNN consisted of 64 LIF neurons. A single dense layer consisting also of 64 LIF neurons was used to encode the integer labels as spike trains with a duration of $100\,\mathrm{ms}$ per label.

We tuned the hyper-parameters using a held-out validation set of 172 characters (688 classes with rotations). We trained for 200 epochs with 200 iterations per epoch using mini batches of size 256. The models were trained with Adam [52] using a learning rate of $\mu = 0.001$, that was reduced by $15\,\%$ every 20 epochs. The output of the network was produced by taking the sum of spikes over the last $30\,\mathrm{ms}$ of $\boldsymbol{z}^{\mathrm{value}}$, and by passing that value through a final output layer with weights W^{out} and a softmax. The error was computed when the network produced its prediction after the query and gradients were propagated through all time steps of the computation. Weights were initialized using Glorot uniform initialization [54] with a gain of $\sqrt{2}$. The feedback delay d_{feedback} was set to $1\,\mathrm{ms}$. We applied L_2 activity regularization after an initial training period of one epoch to all spiking neurons of the model (see Section *Spike rate regularization* for details). The regularization factor λ_ρ was set to 1×10^{-6}. Gradients with an L_2-norm larger than 40.0 were normalized to have norm 40.0.

Details to: Cross-modal associations

We used version 1.0.10 of the FSDD data set available on GitHub, and the MNIST data set from the torchvision data set API (we kept the default train-test split of these data sets). We used one CNN to encode the audio input and one CNN to encode the MNIST images. We extracted 20 MFCCs from the audio signal before feeding it to the convolutional encoder network (we computed 21 MFCCs, but we discarded the constant offset coefficient, that is, the 0th coefficient; MFCCs were zero padded to 30 time samples and scaled such that each coefficient dimension had zero mean and unit variance). The CNNs consisted of four weight layers and had the following structure: $4 \times$ (Conv2D $\rightarrow$ Activation $\rightarrow$ MaxPooling) $\rightarrow$ Flatten. Each block comprises a 64-filter convolution with a kernel size of 3×3 (stride = 1, zero-padding = 1 for each dimension) without bias terms. We used a 2×2 pool size in the max pooling layers. When applied to the 30×20 MFCC features, to the 28×28 MNIST images, this architecture results in a 64-dimensional output space. These CNNs were respectively pre-trained on FSDD/MNIST 10-class digit classification tasks with ReLU functions used as activations after convolutions (i.e., as a conventional ANN), and additional 64×10 dimensional linear classification layers at the output. Following pre-training we discarded the final classification layers. We converted the pre-trained CNNs into spiking CNNs that uses IF neuron dynamics as post-convolution activations, and used a threshold-balancing algorithm to set layer-wise firing thresholds [30], [31] (detailed in Section *Converting pre-trained CNNs*). The converted spiking CNN synaptic weights were later fine-tuned during end-to-end training. Input spikes to the CNNs were produced by applying the grayscale values of the MNIST images and MFCC features to 784 and 600 LIF neurons, respectively (we treated these values as constant input current over $100\,\mathrm{ms}$ per input). The final layers of both CNNs consisted of 64 LIF neurons encoding the input stimuli.

We tuned the hyper-parameters using a validation set which contained 1000 sequences. To encourage the model to learn to associate a specific instance of the FSDD data set to a specific instance of the MNIST data set, we generated training examples as follow: We randomly choose three digits between 0 and 9. We then generated three tuples each containing a randomly drawn instance of these digits as audio file from the FSDD data set, and a randomly drawn instance of these digits as image from the MNIST data set. The query was another randomly drawn sample from one of the FSDD classes in the sequence. Test examples were generated as described in the main text. The image reconstruction was produced by a two-layer fully-connected network with 256 and 784 LIF neurons, respectively. The image was reconstructed by taking the sum of spikes of each neuron in the last layer, scaling the result by $\frac{1}{15}$, and reshaping it to 28×28. We trained our model for 4680 iterations using mini batches of size 256. The models were trained with Adam [52] using a learning rate of $\mu = 0.001$, that was reduced by $15\,\%$ every 780 iterations. During training, all network weights are jointly learned by minimizing the mean squared error (MSE) between the output of the model and the target MNIST image. The error was computed when the network produced its output after the query and gradients were propagated through all time steps of the computation. Weights were initialized using Glorot uniform initialization [54] with a gain

of $\sqrt{2}$. The feedback delay d_{feedback} was set to $1\,\text{ms}$. Starting from iteration 0, we applied L_2 activity regularization to all spiking neurons of the model (see Section *Spike rate regularization* for details). The regularization factor λ_ρ was set to 1×10^{-7}. Gradients with an L_2-norm larger than 40.0 were normalized to have norm 40.0.

Details to: Question Answering

We used version 1.2 of the bAbI data set (we kept the default train-test split of the data set). We used a learned encoding (LE) for sentences as proposed in [34]. The encoding is given by $\boldsymbol{e}_m = \sum_j \boldsymbol{f}_j \circ A\boldsymbol{w}_{m,j}$, where $\boldsymbol{w}_{m,j}$ is a word of a sentence $x_m = \{\boldsymbol{w}_{m,1}, \boldsymbol{w}_{m,2}, \ldots, \boldsymbol{w}_{m,J}\}$ and where A is the embedding matrix and $\circ$ denotes the Hadamard product. The vectors $\boldsymbol{f}_j$ were constant across time steps and were trained jointly with the other parameters of our model. We treated $\boldsymbol{e}_m$ as constant input current, applied for $100\,\text{ms}$ to a layer of 80 LIF neurons, to produces input spikes to our model. We tuned the hyper-parameters on a held-out validation set which was $10\,\%$ of the training set. The networks were trained for 200 epochs on $10\,000$ examples per task with a batch size of 256. The models were trained with Adam [52] using a learning rate of $\mu = 0.003$, that was reduced by $15\,\%$ every 20 epochs. The output of the network was produced by taking the sum of spikes over the last $30\,\text{ms}$ of $\boldsymbol{z}^{\text{value}}$, and by passing that value through a final output layer with weights W^{out} and a softmax. The error was computed when the network produced its prediction after the query and gradients were propagated through all time steps of the computation. Weights were initialized using Glorot uniform initialization [54] with a gain of $\sqrt{2}$. We applied L_2 activity regularization after an initial training period of one epoch to all spiking neurons of the model (see Section *Spike rate regularization* for details). The regularization factor λ_ρ was set to 1×10^{-5}. Gradients with an L_2-norm larger than 40.0 were normalized to have norm 40.0. Since the number of sentences and the number of words per sentence varied within and between tasks, a null symbol was used to pad them to a fixed size. The embedding of the null symbol was constraint to be zero.

Details to: Reinforcement Learning

We used a dense layer consisting of 80 LIF neurons as input encoder in this task. Agents were trained with PPO [37] in actor-critic style using 64 asynchronous vectorized environments. We trained for 4000 iterations. In each iteration, we performed 10 steps (100 steps for 6-card games) in each environment. The surrogate loss was then updated over 4 epochs and 16 training mini batches using Adam [52] with a learning rate of $\mu = 0.0003$. We used a value function coefficient of 0.1, an entropy coefficient of 0.01, and a discount factor of 0.9. The PPO clip parameter was set to 0.2. The agent received a reward of 25 for finding a matching pair and a small penalty of 0.5 for each card flip. The feedback delay d_{feedback} was set to $1\,\text{ms}$. Gradients with an L_2-norm larger than 0.5 were normalized to have norm 0.5. Actions were produced by taking the sum of spikes over the last $30\,\text{ms}$ of $\boldsymbol{z}^{\text{value}}$, and by passing that value through a three-layer fully-connected network with 100, 100, and 4 (6 for 6-card games) neurons, respectively. We used hyperbolic tangent (tanh) activation functions except for the last layer. The critic was also a three-layer fully-connected network comprising of 100, 100, and 1 neurons, respectively. Again, we used tanh activation functions except for the last layer. The critic network received as input, in addition to the sum of spikes over the last $30\,\text{ms}$ of $\boldsymbol{z}^{\text{value}}$, also the agent's observations. Actor and critic networks were initialized using orthogonal initialization [55] with a gain of $\sqrt{2}$. Biases in these networks were zero initialized. Other weights in our model were initialized using Glorot uniform initialization [54] with a gain of $\sqrt{2}$.

Table S1: Sample stories and questions from the bAbI data set [17].

Task 1: Single Supporting Fact

Mary went to the bathroom.
John moved to the hallway.
Mary travelled to the office.
Q: Where is Mary?
A: office

Task 2: Two Supporting Facts

John is in the playground.
John picked up the football.
Bob went to the kitchen.
Q: Where is the football?
A: playground

Task 3: Three Supporting Facts

John picked up the apple.
John went to the office.
John went to the kitchen.
John dropped the apple.
Q: Where was the apple before the kitchen?
A: office

Task 4: Two Argument Relations

The office is north of the bedroom.
The bedroom is north of the bathroom.
The kitchen is west of the garden.
Q: What is north of the bedroom?
A: office

Task 5: Three Argument Relations

Mary gave the cake to Fred.
Fred gave the cake to Bill.
Jeff was given the milk by Bill.
Q: Who gave the cake to Fred?
A: Mary

Task 6: Yes/No Questions

John moved to the playground.
Daniel went to the bathroom.
John went back to the hallway.
Q: Is John in the playground?
A: no

Task 7: Counting

Daniel picked up the football.
Daniel dropped the football.
Daniel got the milk.
Daniel took the apple.
Q: How many objects is Daniel holding?
A: two

Task 8: Lists/Sets

Daniel picks up the football.
Daniel drops the newspaper.
Daniel picks up the milk.
John took the apple.
Q: What is Daniel holding?
A: milk, football

Task 9: Simple Negation

Sandra travelled to the office.
Fred is no longer in the office.
Q: Is Fred in the office?
A: no
Q: Is Sandra in the office?
A: yes

Task 10: Indefinite Knowledge

John is either in the classroom or the garden.
Sandra is in the garden.
Q: Is John in the classroom?
A: maybe
Q: Is John in the office?
A: no

Task 11: Basic Coreference

Daniel was in the kitchen.
Then he went to the studio.
Sandra was in the office.
Q: Where is Daniel?
A: studio

Task 12: Conjunction

Mary and Jeff went to the kitchen.
Then Jeff went to the park.
Q: Where is Mary?
A: kitchen
Q: Where is Jeff?
A: park

Task 13: Compound Coreference

Daniel and Sandra journeyed to the office.
Then they went to the garden.
Sandra and John travelled to the kitchen.
After that they moved to the hallway.
Q: Where is Daniel?
A: garden

Task 14: Time Reasoning

In the afternoon Julie went to the park.
Yesterday Julie was at school.
Julie went to the cinema this evening.
Q: Where did Julie go after the park?
A: cinema
Q: Where was Julie before the park?
A: school

Task 15: Basic Deduction

Sheep are afraid of wolves.
Cats are afraid of dogs.
Mice are afraid of cats.
Gertrude is a sheep.
Q: What is Gertrude afraid of?
A: wolves

Task 16: Basic Induction

Lily is a swan.
Lily is white.
Bernhard is green.
Greg is a swan.
Q: What color is Greg?
A: white

Task 17: Positional Reasoning

The triangle is to the right of the blue square.
The red square is on top of the blue square.
The red sphere is to the right of the blue square.
Q: Is the red sphere to the right of the blue square?
A: yes

Task 18: Size Reasoning

The football fits in the suitcase.
The suitcase fits in the cupboard.
The box is smaller than the football.
Q: Will the box fit in the suitcase?
A: yes

Task 19: Path Finding

The kitchen is north of the hallway.
The bathroom is west of the bedroom.
The den is east of the hallway.
The office is south of the bedroom.
Q: How do you go from the den to the kitchen?
A: west, north

Task 20: Agent's Motivations

John is hungry.
John goes to the kitchen.
John grabbed the apple there.
Daniel is hungry.
Q: Where does Daniel go?
A: kitchen