# Local prediction-learning in high-dimensional spaces enables neural networks to plan

Christoph Stöckl and Wolfgang Maass
Institute of Theoretical Computer Science
Graz University of Technology
stoeckl@tugraz.at, maass@igi.tugraz.at

19th June 2023

Being able to plan a sequence of actions in order to reach a goal, or more generally to solve a problem, is a cornerstone of higher brain function. But compelling models which explain how the brain can achieve that are missing. We show that local synaptic plasticity enables a neural network to create high-dimensional representations of actions and sensory inputs so that they encode salient information about their relationship. In fact, it can create a cognitive map that reduces planning to a simple geometric problem in a high-dimensional space that can easily be solved by a neural network. This method also explains how self-supervised learning enables a neural network to control a complex muscle system so that it can handle locomotion challenges that never occurred during learning. The underlying learning strategy bears some similarity to self-attention networks (Transformers). But it does not require backpropagation of errors or giant datasets for learning. Hence it is suitable for implementation in highly energy-efficient neuromorphic hardware.

## Introduction

Planning and problem solving are at the heart of brain intelligence and AI (Russell and Norvig, 2020). Already the rodent brain is able to solve various planning tasks (Pfeiffer and Foster, 2013), but we do not know how it does that (Mattar and Lengyel, 2022). Many planning and problem solving tasks can be reduced to the problem of finding a short path to a given node in an abstract graph. Powerful offline algorithms such as Dijkstra's algorithm (see supplements Alg. 1), which is referred to as uniform-cost search in AI (Russell and Norvig, 2020) and the computationally more efficient heuristic A* algorithm are commonly used to solve this task by a digital computer. But these algorithms require complete a-priori knowledge of the entire graph, i.e., they cannot learn to operate and plan in a new environment. Furthermore, they do not support real-time decision-making since they first have to compute a complete trajectory to the goal before they propose the first action.

Furthermore, they are completely inflexible and need to restart their computation from scratch when the starting point or the goal is modified, or if an obstacle is encountered on the way. Hence it is unlikely that the brain employs similar methods for planning and problem solving.

On the other hand, the brain has other resources which these AI methods fail to exploit. For example in can represent salient information in very high dimensional coding spaces since it has so many neurons. Special coding opportunities that this high dimensionality provides have already been studied for several decades in the form of operations on high-dimensional vectors, starting with (Plate, 1995). Nowadays this work is commonly subsumed under the term vector symbolic architectures (VSAs) (Kleyko, Davies et al., 2022; Kleyko, D. Rachkovskij et al., 2023). Work on VSAs has attained substantial attention in recent years since it can be efficiently implemented in energy-efficient neuromorphic hardware for AI (Kleyko, Davies et al., 2022; Kleyko, D. Rachkovskij et al., 2023). However, similarly as graph search methods, VSA methods are not based on learning and require instead clever construction. We show that some of the benefits of these high-dimensional codes can also be attained through local synaptic plasticity in a neural network. This can be done in such a way that a key property of VSAs is preserved: Although high-dimensional codes look like random vectors, one can infer from them information about their relationship.

An important hint for the way how planning is implemented in the brain resulted from work on so-called cognitive maps (Tolman, 1948; Behrens et al., 2018; Whittington, McCaffary et al., 2022). This term emerged from the analysis of the representation of landmarks and pathways in 2D mazes in the rodent brain (E. I. Moser, Kropff, M.-B. Moser et al., 2008). More recent experimental data suggest that these cognitive maps also encode information about specific actions that lead from one location to another (Shelley and Nitz, 2021; L. Green et al., 2022). Furthermore, it has been shown that the human brain applies similar cognitive maps for moving and planning in abstract high dimensional concept spaces (Behrens et al., 2018). But it has remained an open problem how such cognitive maps can be learnt.

An inspiring suggestion for that was given in (Buzsáki, 2019): Internal representations of sensory inputs, which have initially no meaning to an organism, acquire a meaning through self-supervised learning of the way how they change through self-initiated actions, e.g. in the course of play. This idea builds on a long history of models in cognitive science and AI, see section 3.5 and 3.6 of (Summerfield, 2022) and (Pezzulo et al., 2007) for reviews. But it had turned out to be difficult to turn these concepts into a functional model for planning and problem solving. We propose here a concrete solution to this challenge, a Cognitive Map Learner (CML). Its self-supervised learning can be implemented through simple local rules for synaptic plasticity that are consistent with experimental data.

We evaluate the capabilities that CMLs can attain through self-supervised learning first on generic planning and problem solving tasks that can be formulated as finding a short path to a given node in an abstract graph. We then apply the CML to 2D navigation tasks, in particular in order to test whether the CML can efficiently exploit underlying regularities of 2D geometry in order to infer the structure of unseen parts of a 2D maze. Finally, we test the generalization capability of the CML in a completely different application domain: Motor control of a quadruped, with the challenge to solve locomotion tasks that never occurred during learning. We conclude with an analysis of similarities and differences between the CML and Transformers.

## Results

## Design principles of the cognitive map learner (CML)

Both observations from the environment and codes for actions are embedded into a high-D continuous space S, see Fig.1a, in which the CML will build a cognitive map (Behrens et al., 2018) through self-supervised learning. For simplicity, we will refer to S simply as state space. Like in the traditional MDP (Markov decision process) framework for reinforcement learning these states will be Markovian. But unlike the MDP framework, actions are embedded into the same space, with the goal that after learning they will denote differences between states that are caused by the action. We will discuss here only the simplest case where the embeddings $Q$ and $V$ are linear functions. The weights of the synaptic connections that represent the matrices that define these embeddings $Q$ and $V$ are learnt through local synaptic plasticity rules in a self-supervised manner. Both embeddings are initialized as random matrices, thereby starting out by mapping observations and actions onto independent random vectors. Note that the capabilities of these linear embeddings can be substantially enhanced if they are preceded by a fixed nonlinear preprocessor, which can assume the role of a kernel, a liquid (Maass, 2011), or a reservoir (Tanaka et al., 2019).

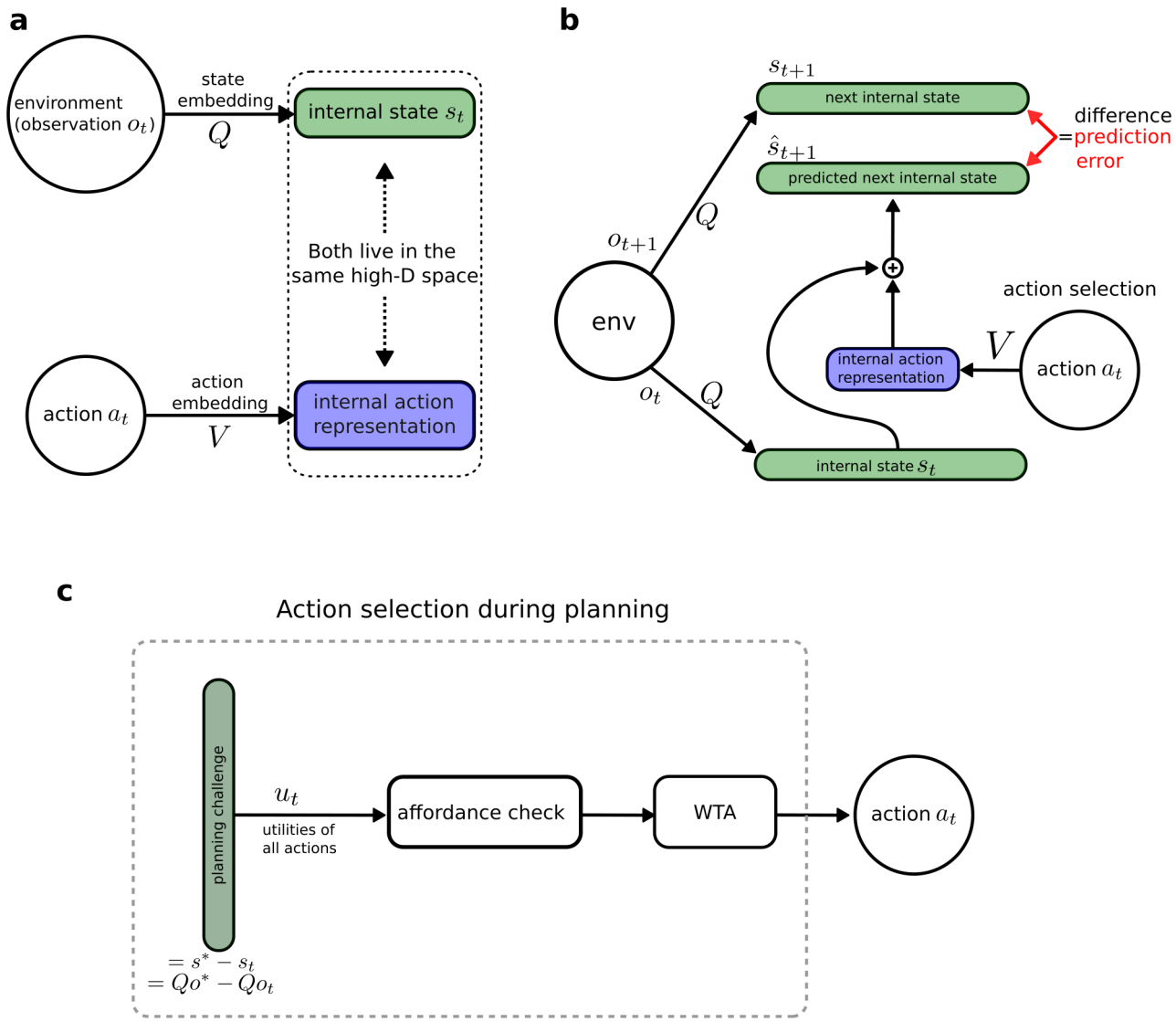


Figure 1: **Scheme for learning and planning of the CML a** Both observations from the environment -that may also encode the current configuration of the agent-, as well as all possible actions are linearly embedded into a common high-D space. **b** Learning of the CML aims at reducing the prediction error for the next state $s_{t+1}$, i.e., the difference between $s_t + Va_t$ and $s_{t+1}$, through synaptic plasticity for the elements of the embeddings $Q$ and $V$. **c** While action selection can be random during initial exploration, the learnt matrix V can be used at any time to assign for any given target observation $o^*$ an utility map $u_t$ for all possible actions. Among those that can be executed in the current state (application of an affordance check) that one with the largest utility value (using a Winner-Take-All (WTA) function) is chosen as next action $a_t$.

The driving force for this self-supervised learning is:

**Principle I: Predictive coding goal:**

$$Qo_{t+1} \approx Qo_t + Va_t \tag{1}$$

**if action $a_t$ is carried out at time step $t$, and $o_t$ is the observation at time step $t$.**

For simplicity, we often write the current state embedding $Qo_t$ as $s_t$ and $\hat{s}_{t+1}$ for the predicted next state $Qo_t + Va_t$. Then the simple local synaptic plasticity rules for prediction learning can be written as

$$\Delta V(t) = \eta_v \cdot (s_t - \hat{s}_t) a_{t-1}^T \tag{2}$$

$$\Delta Q(t) = \eta_q \cdot (\hat{s}_t - s_t) o_t^T, \tag{3}$$

where $\eta_q$ and $\eta_v$ are the learning rates. We refer to Fig.1b for an illustration. These plasticity rules aim to optimize the state- and action embeddings to satisfy Principle I. One can support fast adaptation to new actions by making the learning rate $\eta_v$ for $V$ larger than the learning rate $\eta_q$ for $Q$.

These plasticity rules are instances of the well-known delta rule (Hertz, Krogh and Palmer, 2018) for local learning in neural networks. They are closely related to the BCM (Bienenstock, Cooper, Monroe) rule for synaptic plasticity. The BCM rule has been shown to model several synaptic plasticity processes in the brain quite well (Cooper and Bear, 2012), and is closely related to STDP (spike timing-dependent plasticity) in spiking neural networks (Izhikevich and Desai, 2003; Toyoizumi et al., 2005). Self-supervised online learning by these plasticity rules during exploration can be complemented by letting them also run during replay of episodic memories in subsequent offline phases, as observed in the brain (Diba and Buzsáki, 2007), and also during subsequent planning.

Numerous benefits of predictive coding, a conjectured key feature of brain function, have already been elucidated (R. P. Rao and Ballard, 1999; Friston, 2018; Keller and Mrsic-Flogel, 2018). We show here that it also facilitates planning. More precisely, the plasticity rules (equ. 2 and 3) simultaneously strive to satisfy also the following principle:

**Principle II: An estimate of the utility of an action $a$ for reaching some arbitrary goal $Qo^*$ from the current state $Qo_t$ is provided by the scalar product:**

$$(Va)^T (Qo^* - Qo_t). \tag{4}$$

This Principle II requires that one the space $s$ of state- and action embeddings is sufficiently high-dimensional. It is then directly implied by Principle I for the special case where the target observation $o^*$ is the observation $o_{t+1}$ that arises after executing an action $a_t$, provided that it is applied to vectors in a high-D space that are largely random. The reason is that the term $(Qo_{t+1} - Qo_t)$ is then approximately equal to $Va_t$ according to Principle I. Furthermore, its scalar product with $Va_t$ is substantially larger than with $Va$ for any other action $a$, provided that all action embeddings are approximately orthogonal. This can easily be achieved in sufficiently high dimensional spaces, since randomly chosen vectors

are there with high probability approximately orthogonal. Therefore action embeddings V are best initialized as random vectors: If actions $a$ are encoded by one-hot binary vectors it suffices that the matrix $V$ is initialized with random values from a Gaussian distribution. The embeddings of actions are defined for one-hot action encodings by the column of the matrix $V$, and these columns start out to be approximately orthogonal to each other. It will be shown in the next section (see Fig. 2c) that this orthogonality is in general preserved during learning.

Principle II would fail if different actions are mapped to vectors of different lengths in the state space, because then actions $a_t$ that are mapped to longer vectors would by unduly favored by equ. 4. Therefore a weight normalization is applied to each column of the matrix $V$, i.e., for the incoming weights of a corresponding neuron in a neural network implementation:

$$V(i,j) = \frac{V(i,j)}{\sqrt{\sum_{i'} V(i',j)^2}}. \tag{5}$$

This local normalization ensures that the scalar product in equ.4 is governed by the direction rather than the length of the embeddings of different actions. If these embeddings are in addition approximately orthogonal, the length of a sum of $k$ of them grows according to the law of Pythagoras with the square root of k. Hence the iterated application of Principle I induces an action-dependent metric for the cognitive map: The distance between the embeddings of two states becomes indicative of the number of actions that are needed in order to move from one to the other. This can be clearly seen in the subsequently discussed applications to navigation in general graphs.

Importantly, Principle II remains valid when a sequence of several actions is needed to reach a given target observation $o^*$ from the current observation $o_t$. For example, if a sequence of two actions $a$ and $a'$ is needed to reach it, then according to Principle I we have $Qo^* \approx Qo_t + Va + Va'$. Hence the two actions $a$ and $a'$ can be easily inferred: One computes by computing their utility by taking the scalar product of their V-embeddings with the state difference $Qo^* - Qo_t$ . More precisely, both actions $a$ and $a'$ will get high utility assigned, whereas all other actions will get very low utility since they are approximately orthogonal. A simple affordance check can then be used to decide which of the two actions $a$ and $a'$ should be applied first. This method works also if a larger number of actions is needed to reach the target observation. It relies on a general property of random vectors in high-D spaces, which is also used to support unbinding operations in VSAs: Any finite sum of such vectors has a large scalar product with any individual one of them, since they are approximately orthogonal to each other. In contrast, the scalar product of the sum with any other random vector tends to vanish.

The resulting utilities $u_t$ for all possible actions that are computed by equ. 4 can then be weighted by an affordance module that produces an estimate $g_t$ of the current feasibility of the action, and possibly also its cost or other modulatory factors that just depend on local contingencies and do not need to take into account the current goal. The product element-wise produce of the utility vector $u_t$ and the affordance vector $g_t$ yields the vector $e_t$ of the current eligibity of each possible action

$$e_t = u_t \odot g_t, \tag{6}$$

where the $\odot$ operator denotes the Hadamard product (element-wise multiplication).

The CML then selects a single action $a_t$ from the resulting eligibilities, for example by the WTA (Winner-Take-All) operation,

$$a_t = \mathrm{WTA}(e_t), \tag{7}$$

see Fig.1c for an illustration.

During learning the CML can select actions randomly, through some clever exploration strategy, or while trying to solve the first planning tasks. Continued learning during planning has the advantage that state predictions become especially good in those parts of the state space that are most frequently visited during planning actually occurring planning tasks. A dedicated prior learning or exploration phase is only needed to make sure that a sufficiently large number of states are visited and a fair number of actions are tried out in each of them. In most of our demonstrations we simply used a random exploration strategy.

The review of planning methods in (Mattar and Lengyel, 2022) introduces a distinction between offline and online planning. Characteristic for the latter is that all prior planning results can be discarded once the next action has been chosen. The CML is an online planning method according to this definition, whereas Dijstra and A* are offline methods. The CML is in fact an online planner in a very strict sense, since it does not even compute any further parts of a path in order to select the next action. Therefore it can be referred to as a real-time planner, since its compute time for selecting the next action does not even depend on the size or complexity of the graph in which it plans. In addition, the CML exhibits an astounding flexibility that neither Dijstra nor A* can provide: If it turns out that some action is currently not available, or if the goal is modified, the CML can instantly respond to it without requiring new planning or additional computation time. Surprisingly, the length of paths to a given goal that results from this heuristic real-time planning approach of the CML is often close to optimal. Furthermore, in contrast to the Dijstra algorithm and A*, for planning in physical environments the CML is able to infer the geometry of the environment and can therefore use shortcuts that it never encountered during exploration.

## Application of the CML to generic planning and problem solving tasks

Many general problem solving tasks, such as for example the famous Tower-of-Hanoi task, can be formalized as the task to find a path to a given node in an abstract graph. The directed edges of the graph represent possible actions, and its nodes represents possible intermediate states or goals of the problem solving process. We consider the general case where no insight into the structure of the graph is provided a-priori, so that all of its properties have to be learnt. We first consider the case of a random graph, and then some graphs that pose particular challenges to an online planner

We test the efficiency of planning of the CML with the learnt embeddings $Q$ and $V$ for randomly selected start and goal nodes, or in other words, for generic problem solving. We evaluate the planning performance of the CML by comparing the lengths of the paths from start to goal that it produces with the shortest possible paths that are produced by Dijkstra's algorithm. Note that a comparison of the performance of the CML with the

Dijstra algorithm is a bit unfair to the CML, since the Dijstra algorithm and other search methods that are commonly used in AI, see (Russell and Norvig, 2020), receive complete knowledge of the graph without any effort, whereas the CML has to learn how to operate in it and build its own model of the environment. Furthermore, the Dijkstra algorithm and A* are offline planning methods, whereas CMLs carry out online planning according to the definition of (Mattar and Lengyel, 2022). In fact, a CML chooses the next action in real-time, i.e., within a fixed computation time that does not depend on the size of the graph, the number of paths from which it has to choose, or how far the target node is away. It also does not need extra computing time when an action becomes unavailable, or if the goal suddenly changes. In contrast, Dijkstra's algorithm and A* need to compute a complete trajectory from the current node to the goal before they can produce the next action. In addition, A* has to run again from scratch if the start or goal node changes. Dijkstra's algorithm has to do that also if the start node changes.

### Generic random graphs

We focus first on the example of a randomly connected graph with 32 nodes, where every node has a random number between two and five edges to other nodes. The edges of the graph are undirected since they can be traversed in any direction. But each traversal of an edge in a given direction is viewed as a separate action, hence there are two actions for every edge. The random graph is visualized in Fig. 2a through a t-SNE projection to 2D of the initial representation of its states through random vectors in a 200-dimensional space.

We use an extremely simple exploration approach where random actions are chosen used during learning. More precisely, the CML takes 200 random walks of length 32 from randomly chosen starting points. These random walks are subsequently replayed while learning of the embeddings $Q$ and $V$ through the synaptic plasticity rules (equ. 2 and 3) continues. Fig. 2b shows a corresponding 2D projection of the states which the CML has constructed during this learning process. Most pairs of nodes that are connected by an edge have now the same distance from each other. This results from the fact that the CML has learnt to approximate their difference by the embedding of the action that corresponds to the traversal of this edge. These action embeddings all have the same length because of the local normalization of the embedding matrix $V$ according to equ. 5. One clearly sees from even from the 2D projection of the geometry of the learnt state space that in order to reach any given node one from one given starting point one can use any edge from the start node that roughly points into the direction of the goal node. Hence the learnt embedding of observations and action forms a high dimensional cognitive map whose geometry makes planning easy. The action embeddings are initialized so that they are initially orthogonal, since orthogonality supports action selection during planning according to Principle II. This orthogonality is automatically preserved quite well during the learning process due to the high-dimensionality of the space in which they are embedded, see Fig. 2c and d. Producing a cognitive map with a geometry that approximately satisfies Principle I requires solving a quite difficult constraint satisfaction problem. For example, it requires that the sum of action embeddings of any cycle in the graph sum up to the 0-vector. Fig. 2f shows that this constraint is approximately satisfied by the CML through its local synaptic plasticity (compare the norm of the sums of action embeddings along a cycle with the length of the embedding of any single action, which has value 1). Fig. 2e shows that refined adjustments to the cognitive map through synaptic plasticity that occurs after state prediction works already quite well (blue curve) provide additional improvement of the planning performance

(red curve). Fig. 2g shows an example for the continual adjustment of utility estimated for each action while a plan is executed. Note that several alternative next actions have high utility at most steps of the execution, thereby reducing reliance on a single optimal path, as proposed by offline planning methods, and providing inherent flexibility to the online planning process.

The CML produces for this graph a trajectory from an arbitrary start node to an arbitrarily given target node using an average of **3.515** actions (std. deviation 1.425). This is very close to the average shortest possible path of 3.464 (std. deviation 1.352), which the Dijkstra algorithm produces.

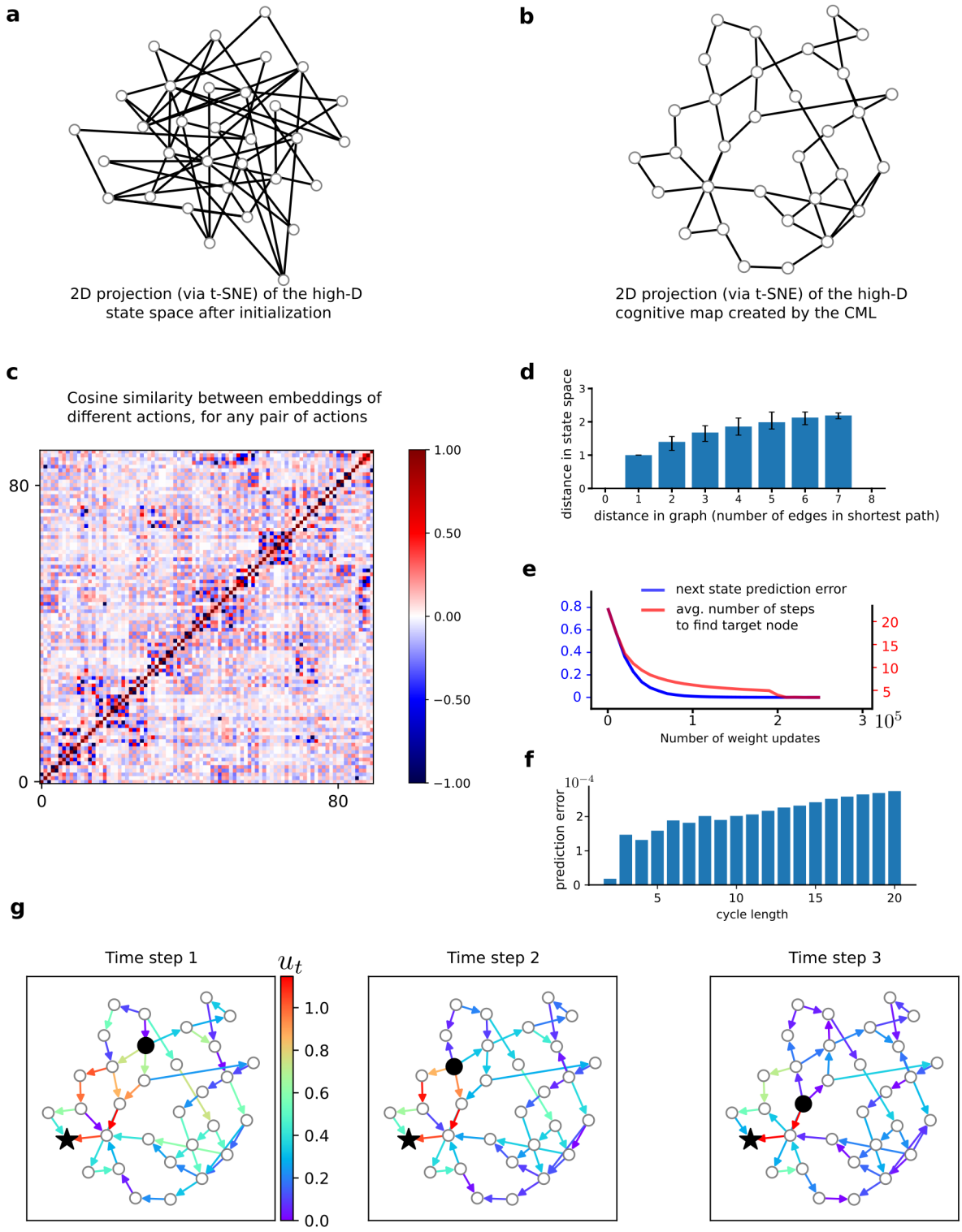
a
2D projection (via t-SNE) of the high-D state space after initialization
b
2D projection (via t-SNE) of the high-D cognitive map created by the CML
c
Cosine similarity between embeddings of different actions, for any pair of actions
80
0
0
80
1.00
0.50
0.00
−0.50
−1.00
d
distance in state space
distance in graph (number of edges in shortest path)
e
next state prediction error
avg. number of steps to find target node
Number of weight updates
$10^5$
f
prediction error
$10^{-4}$
cycle length
g
Time step 1
Time step 2
Time step 3
$u_t$

Figure 2: **Properties of the cognitive maps which the CML builds through self-supervised learning a** 2D-projection (via t-SNE) of the initial state embedding of a generic random graph. **b** 2D projection of the state embedding for the same graph after learning. One sees that the emergent geometry of the cognitive map supports planning a path from any start node to any goal node through a simple geometric heuristic: Take an edge that roughly points into the direction of the goal node. **c** Cosine similarity of embeddings of different actions under the learnt embedding $V$. One sees that most action embeddings are orthogonal to each other, with some exception for actions that start at the same node (these are adjacent in the chosen order of all actions for this plot). **d** The average distance between any pair of nodes in the learnt cognitive map can be approximated by the square root of the length of the shortest path that connects them, as expected from the law of Pythagoras for the case that embeddings of different actions are orthogonal to each other. The error bars indicate two standard deviations of the distribution of distances in state space for a given distance in the abstract graph. **e** Visualization of the learning process of the CML for the given random graph. After a while the Principle I is already almost perfectly satisfied, i.e., no significant prediction errors occur anymore (blue curve). During a bit of further learning finer adjustments arise in the cognitive map so that the online planning by the CML usually finds near-optimal paths to given goals (red line). **f** When one applies Principle I iteratively to the edges of any cycle in the graph, the sum of the embeddings of the corresponding actions should be close to the 0-vector. one sees that this is satisfied quite well for the given random graph. **g** Example for a given start node (black disc) and goal node (black star) in the random graph. Utilities of possible actions are indicated for each step during the execution of a plan, provided they have positive values. One sees that utilities are online adjusted when the current node (black ball) moves. Furthermore, several alternative options are automatically offered at most steps during the execution of a plan.

### Finding the least costly or most rewarded solution of a problem

Different actions incur different costs in many real-world problem solving tasks, and one wants to find the least costly path to a goal, rather than the shortest path. The cost of an action could represent for example the effort or time that it requires, or how much energy it consumes. Not only the human brain is able to provide heuristic solutions to these more challenging problem solving tasks, but also non-human primates (S. J. Green et al., 2020). Hence the question arises how neural networks of the brain can solve these problems. One can formalize them by adding weights (=costs) to the edges of a graph, and searching the graph for the least costly path to a given goal, where the cost of a path is the sum of the costs of the edges that it traverses.

A simple extension of the CML provides heuristic solutions for this type of problems. One can either let the cost of an action regulate the length of its embedding via a modification of the normalization of columns of V according to equ. 5, or one can integrate its cost into the affordance values $g_t$ that are produced by the affordance module. The result for planning will be the same in both cases, since the eligibility of an action is according to equ. 11 the produce of its current affordance and utility. A key difference is that in the first option a cost of an action can modify the whole cognitive map, and thereby support foresight in planning that avoids future costly actions. Integrating cost values into the affordances has the advantage that they can be changed on the fly without requiring new learning. But

costs affect then only the local action selection in a greedy manner. We demonstrate in an example that this simple method supports already planning in weighted graphs very well. We assigned random integer weights (= costs) between 4 and 7 to the edges of the previously considered random graph. Corresponding affordance values $g_t$ are shown in Fig. 3b for the cognitive map which the CML produces after learning. The CML produces for random choices of start and goal nodes in this graph a solution with an average cost of **22.73**. This value is quite close to optimal: The least costly paths that the offline Dijkstra algorithm produces have an average cost of **20.18**.

Also many reinforcement learning problems can be formulated as path planning problems in a weighted graph. In this case the inverse of the weight of an edge encodes the value of the reward that one receives when this edge is traversed. Hence the same CML heuristic as before produces a path from a given start to a given goal that accrues an approximately maximal sum of rewards. In many reinforcement learning tasks one does not have a single goal node, but instead a set of target nodes in which the path from the start node should end. An example is the graph shown in Fig. 3c; note that this graph does not have to be a tree. It encodes the frequently considered case of a 2-step decision process (Russell and Norvig, 2020; Mattar and Lengyel, 2022). The task is here to find a path from the bottom node to a node on layer 2 that accrues the largest sum of rewards. This is equivalent to finding the most rewarded path to a virtual node that is added above them, with added edges from all desirable end nodes. Hence also this type of task can be formalized as the task to find the least costly path from a given start to a given goal node in a weighted graph. Therefore the CML provides approximate solutions also to such multi-step decision processes that are commonly posed as challenges for reinforcement learning (Russell and Norvig, 2020). Note also that the type of reinforcement learning problems that were tackled with the elegant flexible method of (Piray and Daw, 2021) have a fixed set of terminal states. Since there only the transition to a terminal state yields a reward, the task to find a maximally rewarded solution is exactly equivalent to finding a least costly path to an added virtual node which receives weighted edges from all terminal states. Hence these problems can also be solved by the CML. An interesting difference is that the CML does not need to have a default policy, and requires only local synaptic plasticity for learning.

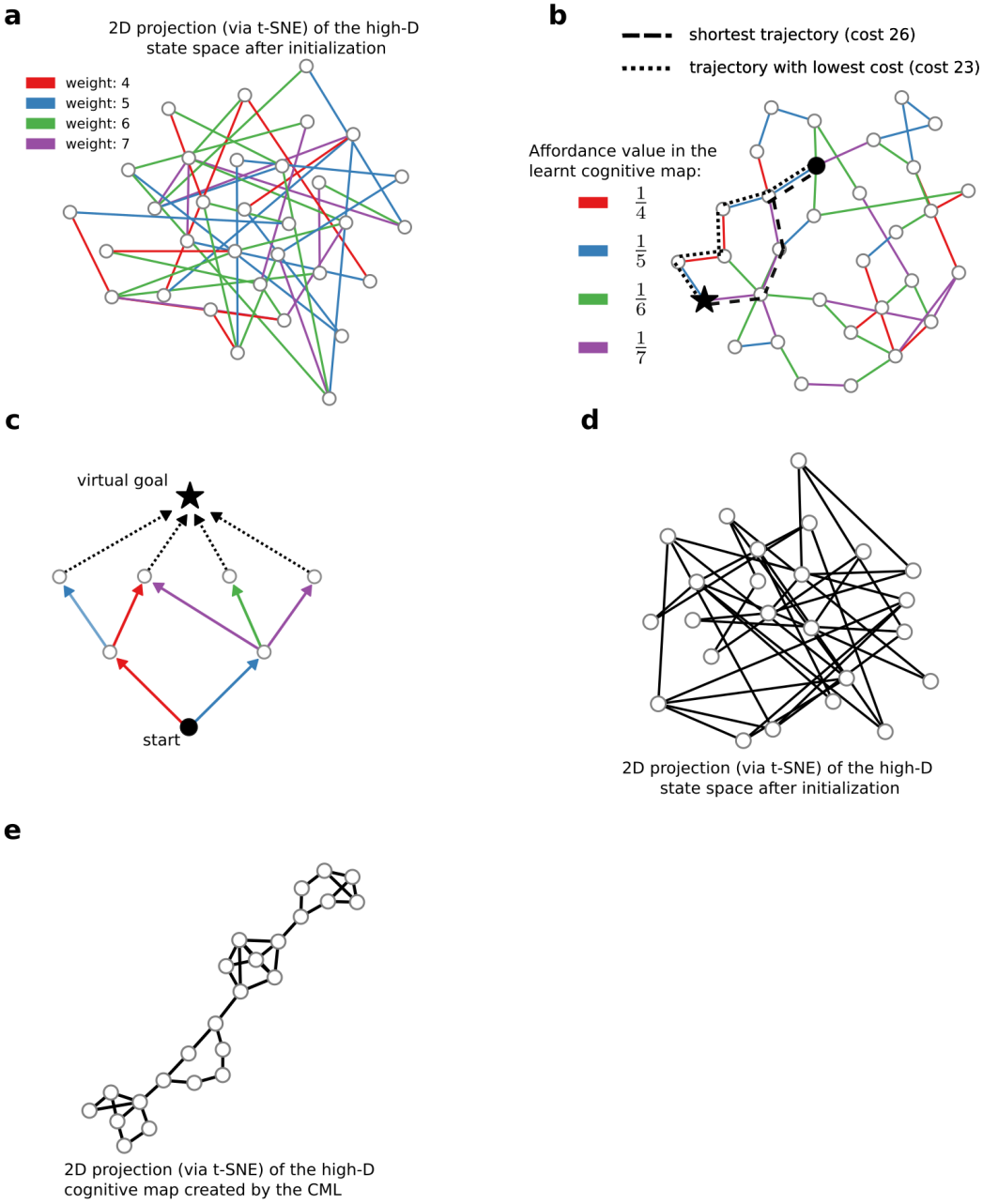
a
2D projection (via t-SNE) of the high-D state space after initialization
weight: 4
weight: 5
weight: 6
weight: 7
b
shortest trajectory (cost 26)
trajectory with lowest cost (cost 23)
Affordance value in the learnt cognitive map:
$\frac{1}{4}$
$\frac{1}{5}$
$\frac{1}{6}$
$\frac{1}{7}$
c
virtual goal
start
d
2D projection (via t-SNE) of the high-D state space after initialization
e
2D projection (via t-SNE) of the high-D cognitive map created by the CML

Figure 3: **Further problem solving challenges for the CML.** The weights in panel **a** were sampled uniformly from a set of integers containing 4, 5, 6 and 7. The affordance values in panel **b** were obtained by dividing 1 by the weight of the corresponding edge. Since the eligibility of an action can be made proportional to this value, the CML prefers actions for which the corresponding edge in the graph has a lower value. Hence the CML tries to find for given start and goal nodes a path of lowest cost instead of a shortest path. Panel **c** illustrates a simple method for reducing a commonly considered class of reinforcement learning problems, 2-step decision processes, to a task where a path to a given goal node has to be found: One just needs to add a new virtual goal node that receives edges from all admissible end notes of the decision process. Weights, or rather their inverse, may encode in this case the rewards that are given when an edge is traversed. Panel **d** depicts the 2D-projection (via t-SNE) of the initial state embedding of a random small world graph. In panel **e** the 2D projection of the state embedding of the learnt cognitive map is shown for the same graph as in panel d. The small world structure of the graph is clearly revealed in this cognitive map, which resulted entirely from the self-supervised learning of the CML.

Fig. 2g sheds some light onto the inner workings of the CML, where the values of $u_t$ are depicted. Intuitively, the utility value should indicate how useful taking a certain action is for moving toward the target state $s^*$. As can be seen directed edges pointing toward the goal state have a higher utility, especially when they are between the current node and the target node. The utility values for several search tasks on different types of graphs can be found in Fig. S1.

### Graphs that pose special challenges for online path planning

Small world graphs can be seen as special challenges, since one needs to identify the precise that node that allows escape from a local cluster. Therefore we tested the CML on a graph that had four clusters each consisting of 6 densely interconnected nodes, but only a single connection from one cluster to the next. The initial state embedding is shown for this graph in Fig. 2d. The cognitive map which the CML produced for this graph after learning is depicted in Fig. 2e. The CML solved all planning tasks for given pairs of start and goal nodes very efficiently, in fact optimally: It produced in each of 1000 samples of such tasks the optimal solution, with an average path length of **4.652** (std. deviation 2.8). Hence it achieved the same performance as the offline Dijkstra algorithm.

Another concern is that a heuristic problem solver such as the CML has problems if multiple paths of the same length to a goal exist, since dealing with such ambiguity appears to require some global knowledge of the graph structure. But we found that this feature did not impede the performance of the CML for the example graphs that we examined: The CML needed on average **2.477** (std. deviation 1.105) actions to reach the goal (average over 1000 trials), which is the same as the optimum computed using Dijkstra. In other words, the CML found in each trial among many possible closely related solutions the shortest path. A sample solution is shown in Fig. S1a.

A further concern is that a learning based online planner may have problems if there are dead ends in a graph: Paths into these dead ends and paths for backing out of them are traversed during learning, and could potentially become engraved into the cognitive map and cause detours when it is applied to planning. To explore how the CML copes with this difficulty, we designed a graph which contains a large number of dead ends. The graph as

well as the utility values computed by the CML during the execution of a plan can be seen in Fig. S1b. We found that despite the difficulty of having to deal with dead ends, the CML managed to achieve the same performance as Dijkstra on this graph.

## Path planning in physical environments

The previously considered abstract graphs had no symmetries that could enable goal-directed use of an edge without prior exploration of its impact. But if edges represent path segments in a physical environment, its inherent symmetries enable prediction of the impact of an action without ever having tried it. The cognitive map of rodents is in fact able to exploit such symmetries, since rodents were found to be able to integrate short-cuts into their path planning in 2D mazes which they had never encountered during exploration (Gupta et al., 2010). Hence a key question is whether the cognitive maps which the CML produces can also support such powerful generalization of learned information. We tested this in a task that is inspired by the task design in (Ritter et al., 2020) and (Whittington, Muller et al., 2020). In the simplest case the environment is a rectilinear 2D grid of nodes, where each node gives rise to a unique observation which is denoted by an icon in Fig. 4. At each node there are the same 4 actions A, B, C, D available, see Fig. 4a, each of them encoded by the same one-hot code at each node. The CML has initially no information about the meaning of these actions, which were realized during exploration by 1-step movements in the 4 cardinal directions of the grid. The only information that the CML received during learning were the observations that resulted from the execution of an action, starting at different positions in the grid, see Fig. 4b for examples. We allowed 22 exploration sequences, which were all of length 3, and made sure that not all edges of the graph were encountered during learning, as indicated in Fig. 4c. No additional information about the geometry of the environment was provided to the CML, and it had to infer the structure of the 2D maze from accidentally encountering the same observation in different contexts with other observations. We tested the CML after learning especially for start and goal nodes that never occurred both on any exploration sequence, and where in addition the shortest path between them involved unexplored edges, see Fig. 4 d and e for an example. Nevertheless, the CML produced solutions to these planning tasks that had the shortest possible length, and made use of unexplored edges, see Fig. 4e for an example.

The cognitive map which the CML had generated during learning, see Fig. 5a, explains why it was able to do that: The PCA analysis of this high dimensional cognitive map shows that the CML had morphed it into a perfect 2D map of the spatial relations between the observations that it had encountered during learning. An animation illustrating the emergence of its cognitive map during the learning phase can be found here[1]. One sees that the internal representations of observations keep changing until a perfect 2D has been generated. This self-organization is reminiscent of earlier work on self-organized maps (Kohonen, 1990). There the 2D structure of the resulting map resulted from the 2D connectivity structure of the population of neurons that generated this map. In contrast, there is no such bias in the architecture of the CML. In fact, the same CML can create cognitive maps that model very different physical environments, e.g. also 3D environments. Since cognitive maps for 3D environments are hard to visualize, we show instead in Fig. S2 and Fig. S3 an application to a hexagonal 2D environment, where the same generalization

[1]https://drive.google.com/file/d/19ziVY5PqXdbRk6jvUJECnTY7G6aKYxDN/view?usp=sharing

and abstraction capability can be observed as for the rectilinear environment. In Fig. 5b we show the result of applying to the cognitive map which the CML has generated an abstract measure for compositional generalization capability of neural representations: the parallelism score (Ito et al., 2022). The very high parallelism score for the two state differences that are examined there indicates that a linear classifier can discern for any pair of observation which of them is attained from the other by applying a particular action, even for pairs that were not in the training set for the linear classifier.

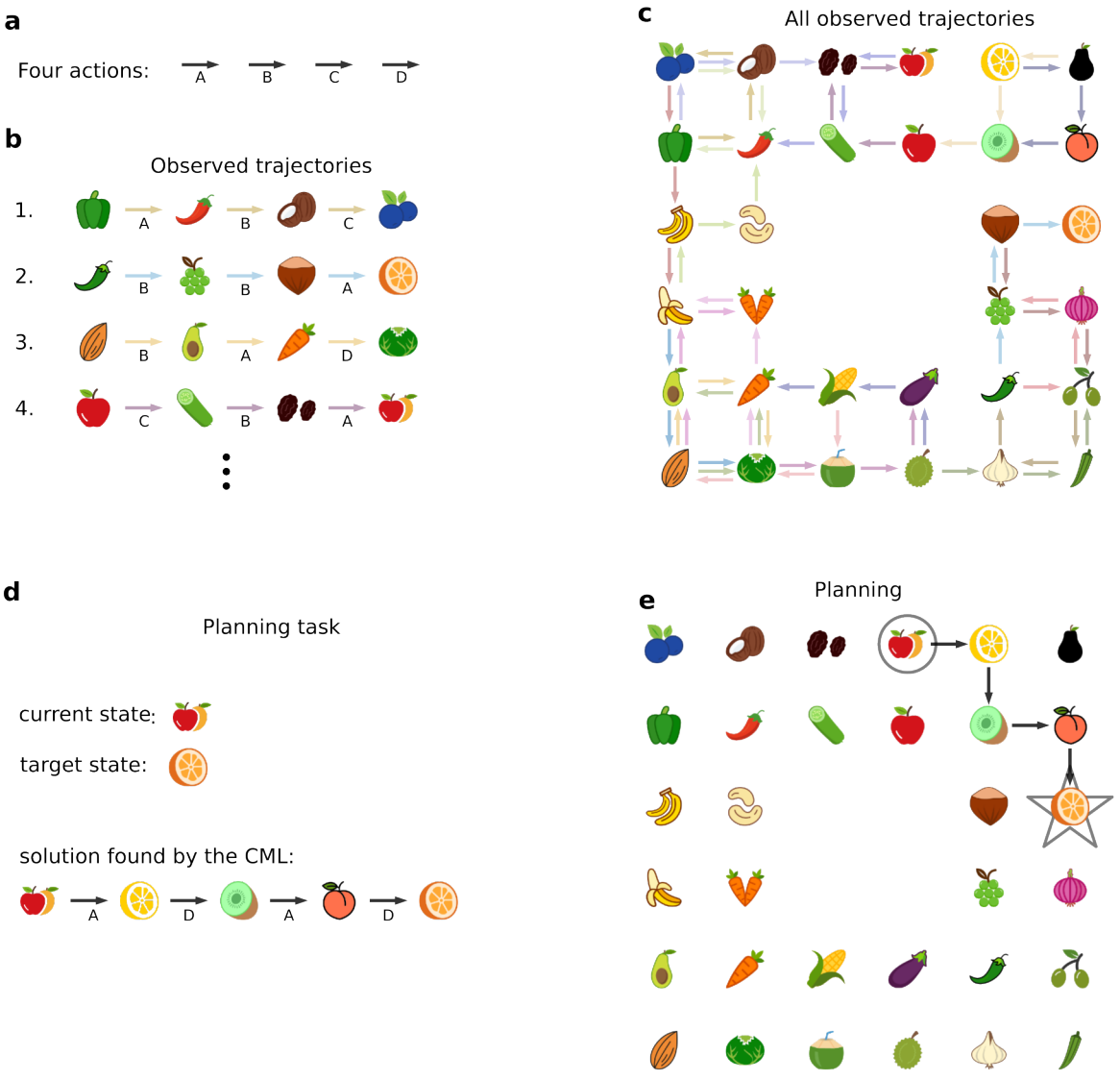

Figure 4: **The CML learns fundamental properties of a 2D environment, and exploits them for generalization. a** Four different actions, which can be applied in any state. The CML has no prior knowledge about the meaning of them. Panel **b** shows samples of the 22 trajectories that the CML has observed during learning from the perspective of the CML: Only relations between actions and observations are provided in short trajectories with three actions. Panel **c** Provides a birds-eye view of all 22 trajectories during learning, Note that some combinations of actions and states were never tried out. Panel **d** provides an example of a planning task. A start observation and a target observation are given to the CML, and it has to produce a sequence of actions which represents the shortest path from start to goal. **e** The solution of the task from panel d which the CML produced. Note that three of its four actions had never been tried out from these states (observations) during exploration (compare with panel c). Hence the CML is able to generalize learnt knowledge and apply it in new scenarios.

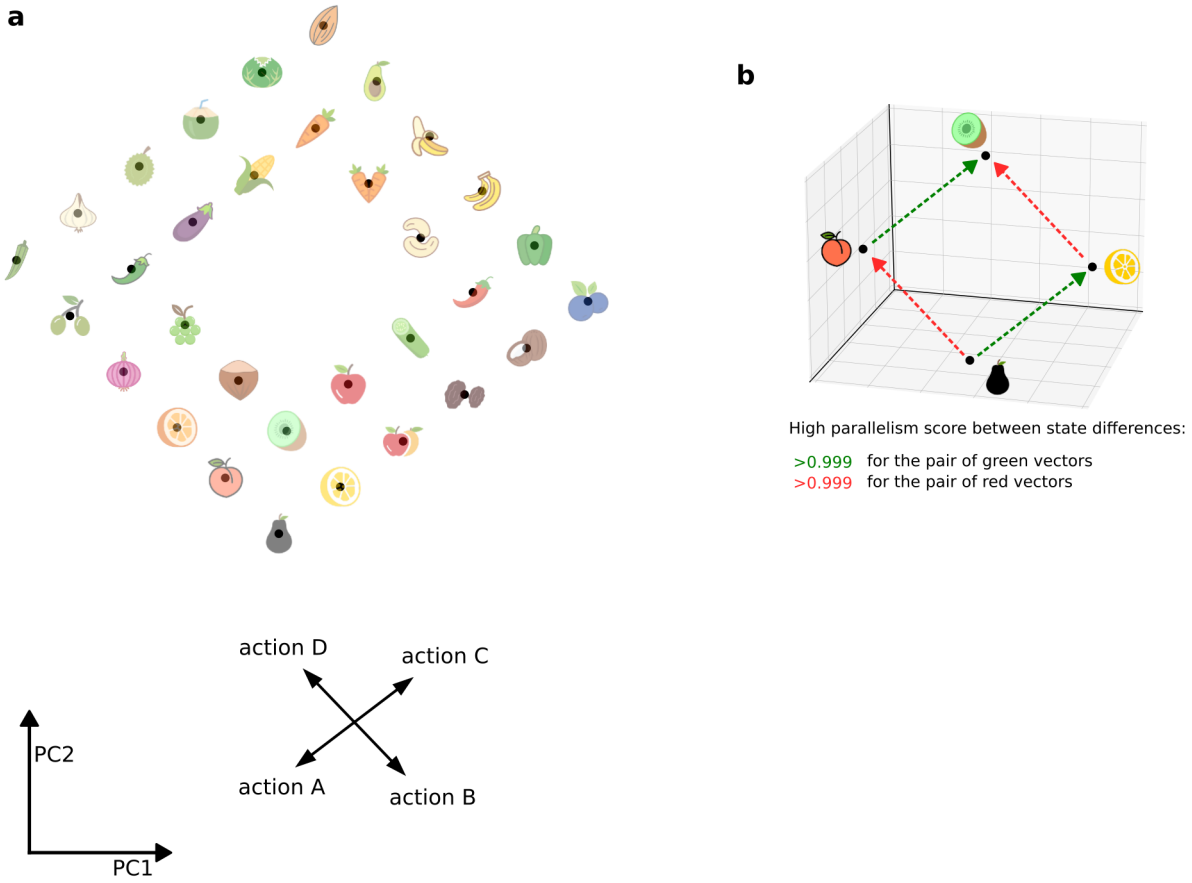


Figure 5: **Structure of the cognitive map which the CML generated during exploration.** The same incomplete exploration of a 2D environment as in the previous figure was used. **a** Projection of the learnt cognitive map to its first two principal components. Additionally, the columns of $V$, i.e., the action embeddings, are shown in the same 2D projection. One sees that the learnt relations between actions and observations perfectly represent the ground truth, thereby explaining why the CML produces optimal shortest paths during planning for each given start and goal observation. Panel **b** shoes the parallelism score of the cognitive map for pairs of states where one results from the other by applying a specific action (indicated by red and green arrows), in two different contexts (= initial states). The parallelism score is the cosine between the state differences that result from applying the same action in two different contexts. According to (Ito et al., 2022) the parallelism score is a useful measure for the compositional generalization capability for a system of neural representations, both in the brain and in artificial neural networks.

## Generalization of knowledge attained through self-supervised learning for controlling a quadruped

Organisms appear to be very good in applying learnt experience from motor control for solving new movement challenges. In contrast, existing learning approaches in robotics are often criticized as being too rigid, and lacking generalization capabilities. Hence we wondered whether the CML could provide also new perspectives for robot learning. The beauty of the CML approach to motor control is that it can instantly be applied to a new task, also in cases where the target state quickly varies over time. We demonstrate this for a standard benchmark task for motor control: Locomotion of a quadruped robot (see Fig. 6a), commonly referred to as ant (in spite of the fact that ants have more than 4 legs). Actions consist of torques that are sent to the 8 joints of its 4 legs. This task was adapted from a popular reinforcement benchmark, (Todorov, Erez and Tassa, 2012; Brockman et al., 2016). While in the original task the goal is to move the ant as fast as possible in the direction of the x-axis, we consider here the more demanding tasks to navigate to a dynamically moving target object, or to escape a predator. This environment provides 29 dimensional observations $o_t$, which contains in particular information about the angles of the 8 joints, their respective speed, the (x,y) coordinates of the center of the torso, and the orientation of the ant, see Tab. 1 for details.

In the learning phase, the CML explores the environment by randomly selecting actions (often referred to as motor babbling), see the video [2]. Importantly, the CML was never optimized or learnt to move to a goal location - it could only observe state changes during 200 trajectories taking random actions. But it could solve this task nevertheless. In the simplest planning challenge, the CML was tasked to navigate the ant to arbitrarily given goal locations in a distance of 20 meters away from the starting position (see video[3]). In this case, the target observation $o^*$ was defined by the desired (x, y) coordinates of the center of the torso, while leaving all other components of the observation the same as in the current observation $o_t$. The resulting trajectories of the ant are depicted in Fig. 6b, where the stars indicate the goal location and the line plotted in the corresponding color indicates the path taken by the center of the ant torso. The CML solved these tasks with a remaining average distance to the target of less than $0.515m$ (average over 1000 trials). Fig. 6c illustrates the correlation between the state prediction error and the performance of the CML in terms of the remaining distance to the target location.

Due to the task agnostic nature of the CML one can use the same CML also to solve a completely different task, without any new learning: Tasks where the goal changes continuously. We considered two such tasks, see Fig. 6d. In one, that arises when chasing prey, the first two coordinates of the target observation $o^*$ change continuously, with its current value indicated by the center of the red ball. In another task, corresponding to fleeing from a predator, the first two coordinates of the target observation are defined by a position 5m away from the current position in the direction away from the current position of the predator (marked by the center of the red square in Fig. 6d). Also these new tasks were handled very well by the same CML, see the online available videos for fleeing from a predator (video [4]) or chasing a target (video[5]).

[2]https://drive.google.com/file/d/13ST3CUxJ6EtDQs-zuazqYH254FF7AC81/view?usp=sharing
[3]https://drive.google.com/file/d/1wzAEW9DFE0NPl1LT2o6XkWmOOInsWTlj/view?usp=sharing
[4]https://drive.google.com/file/d/1t74L6b4zwDf_iC53oaK0ZMIRibViy71r/view?usp=sharing
[5]https://drive.google.com/file/d/1iND1JMzUgNQuTZNRncRq05W9KIqIbC9Y/view?usp=sharing

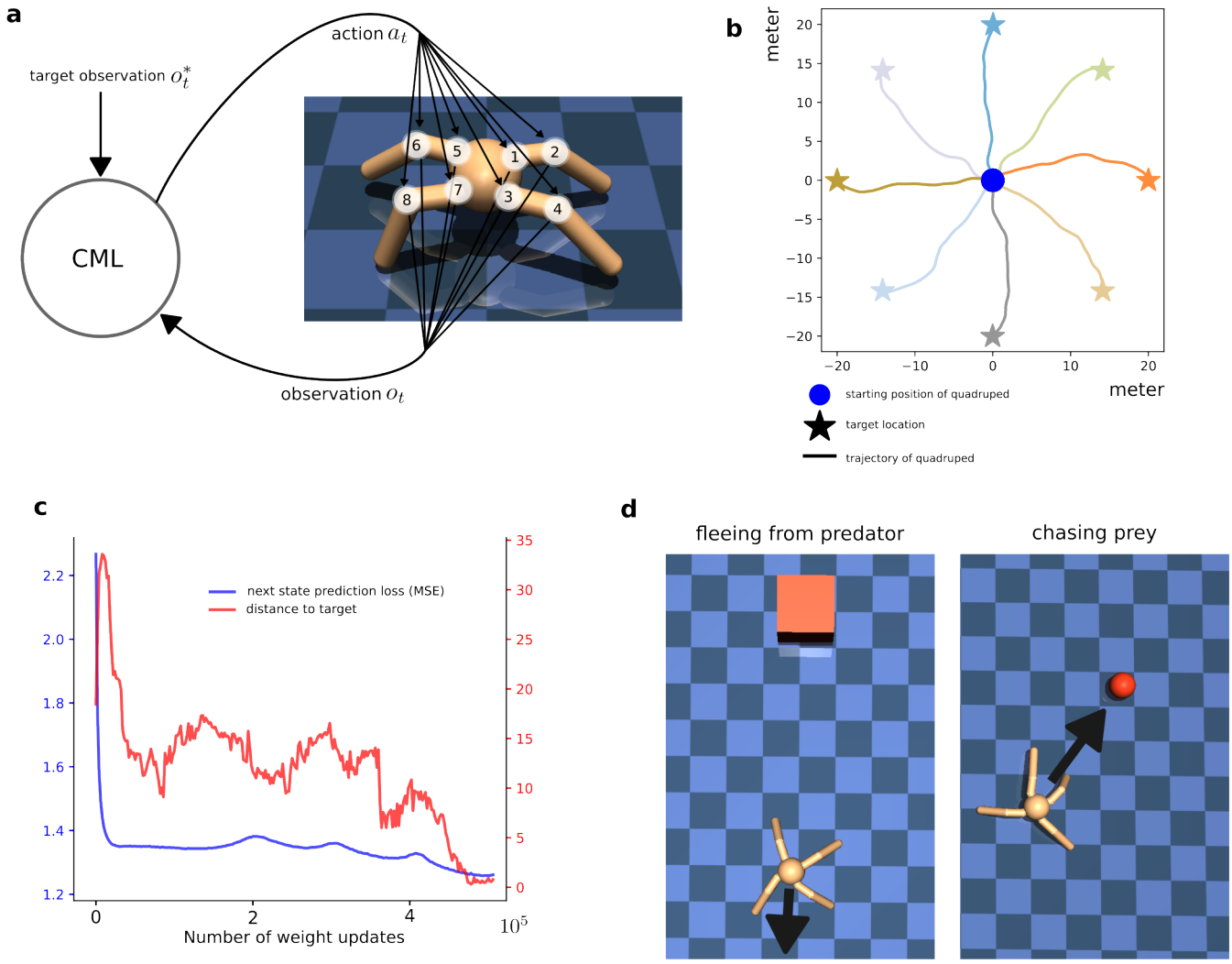


Figure 6: **Application of the CML to motor control.** Panel **a** depicts the quadruped ("ant") with its 8 joints that can be controlled individually. Panel **b** provides first evidence for the generalization capability of the CML controlled. Although it only used random actions ("motor babbling") during learning, it was immediately able to move the ant to a given goal. Resulting paths of the center of the ant torso are shown for 8 sample goal states. The blue circle in the middle of the panel indicates the starting position of the ant, the stars depict 8 different goal states. Trajectories to them which are produced by CML planner were low-pass filtered and plotted in the same color as the goal. The ant was reset to the center location after every episode. Note that no re-learning was required for moving to a different goal. Panel **c** visualizes the evolution of the next state prediction error during learning (blue curve), as well the average distance to the target location after the end of the episode (red curve) when learning was interrupted after every 2000 weight updates the partially learnt cognitive map was applied to the task of panel b, averaging over 32 evaluation trajectories. Panel **d** illustrates two further tasks that can be solved by the same CML without any new learning or instruction: fleeing from a predator(video) and chasing a prey (video).

## Neuromorphic hardware

CMLs are of special interest for implementation of planning, problem solving, and robot control in energy-efficient neuromorphic hardware. Whereas backpropagation and backpropagation through time can not be implemented on-chip on current neuromorphic hardware, the local synaptic plasticity rules that the CML required for learning can be implemented on-chip on Loihi (Davies et al., 2021) and other neuromorphic chips. In view of the surprising capabilities that learning with these simple plasticity rules, the CML appears to be an attractive option for endowing a neuromorphic chip with functionally powerful on-chip learning capabilities.

Note also that the computational operations of the CML (multiplication of vectors with learnt matrices) is well suited for an in-memory computing approach (Verma et al., 2019), where a crossbar of memristors (Strukov et al., 2008; M. Rao et al., 2023) could represent the learnt embedding matrix, and each multiplication of a matrix with a vector can be carried out in one computation step. In addition, both the SpiNNaker approach (Khan et al., 2008) and several more recent energy-efficient neuromorphic hardware designs support the implementation of neurons that emit a package of several bits instead of just spikes, thereby supporting a direct implementation of the CML without first porting it into spiking neural networks. Since neuromorphic hardware implementations commonly provide limited precision for weight values, often limited to eight bits (Davies et al., 2021), we tested whether this could pose a problem for CML a neuromorophic CML implementation. A control experiment was performed for learning to plan for the abstract random graph considered in Fig. 2, where the bit precision of weights was reduced to just 8 bits. We found that this constraint did not have a large impact on the planning performance of the CML. A degeneration of the planning capability was only detected when 6 or fewer bits were available for encoding a synaptic weight.

## Relation of the CML to self-attention approaches (Transformers)

Some aspects of CMLs are reminiscent of the self-attention mechanism described in (Vaswani et al., 2017), the work horse behind the Transformer architecture. There one computes:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V, \tag{8}$$

where $Q$ corresponds to a matrix containing the queries, $K$ to a matrix containing the keys, $V$ a matrix containing the values, and $d_k$ is a scaling constant. During self-attention, the matrices $K$, $Q$ and $V$ are computed directly from a sequence of tokens. The CML on the other hand does not require a sequence of tokens, it rather only considers a single token, which is the target direction in state-space $d_t = s^* - s_t$ and can be interpreted as a single query. This feature arises from the implicit assumptions that observations are Markovian, i.e., only the most recent observation is relevant for making a prediction. As there is no sequence of tokens, unlike the Transformer, the CML considers fixed keys $K$ and values $V$ that do not depend on the current observation or action.

In the CML, the next state prediction $\hat{s}_{t+1}$ can be written as:

$$\hat{s}_{t+1} = s_t + \text{WTA}(g_t \odot K \cdot d_t)^T V^T, \tag{9}$$

where $d_t = s^* - s_t$, which contains information about both the current state $s_t$ and the target state $s^*$ acts as the query, $g_t$ represents the affordance gating values, which indicate the availability of an action in the current position in the environment and the keys $K$ are considered to be equal to the values $V$ in the CML. Equation 9 displays striking structural similarities with the attention mechanism used in Transformers (equ. 8), especially with respect to the relationships among the variables involved, if one takes the previously described differences into account. The latter entail that the attention matrix is not quadratic of dimension $\mathbb{R}^{(n_{\text{seq}}, n_{\text{seq}})}$, where $n_{\text{seq}}$ is the sequence length of the input tokens, as in Transformers. Rather, the attention matrix in the CML has the shape $\mathbb{R}^{(1, n_{\text{seq}})}$, as there is only a single query $d_t$. Note that $n_{\text{seq}}$ corresponds to the number of actions $n_a$ in this analogy. In the CML, the attention matrix measures the utility for every action, assigning a high utility value to actions that are more useful and a lower value for less useful actions. Therefore, one could say that the CML "attends" to useful actions.
We also want to point out that the addition of the current state $s_t$ in equ. 7 is reminiscent of residual connections, that are also widely used in Transformer architectures. This is a common practice in the literature for neural network based approximations of Transformers, see (Ramsauer et al., 2020; Whittington, Warren and Behrens, 2021; Krotov and Hopfield, 2020).

## Discussion

The capability to plan a course of actions, and to instantaneously adjust the plan in the face of unforeseen circumstances is a hallmark of general intelligence. We have shown that a simple neural network model, that we call a cognitive map learner (CML), automatically acquires these capabilities from self-supervised learning to predict the next sensory input (observation) after carrying out an action. We have demonstrated that the CML can learn on its own how to solve a wide range of tasks, even tasks to which it had never been exposed before: General problem solving tasks that are commonly formalized in terms of navigation on abstract graphs (Fig. 2, 3), navigation in par2D environments that were only partially explored (Fig. 4, 5), and goal directed quadruped locomotion from learning during motor babbling (Fig. 6). Surprisingly, the CML approximates for navigation in abstract graphs the performance of best offline planning tools in AI, such as the Dijstra algorithm and the A* algorithm, although it plans in a much more flexible and economical online manner that is able to accommodate ad-hoc changes of the goal or in the environment, e.g. when a particular action is no longer available. We have also shown in Fig. 5b that the cognitive map which the CML generates for a physical environment is especially suited for compositional generalization. According to (Mattar and Lengyel, 2022) it is largely unknown how the brain solves planning tasks. The CML provides a concrete hypothesis for that, since its simple neuronal architecture and local synaptic plasticity rules are within the scope of brain neurophysiology.

The CML is also able to explain and reproduce a characteristic feature of neural codes in the brain that has received increasing attention in recent years: Neural representations of internal states and sensory inputs have been found to drift on the time scale of days and weeks, while task performance remains stable (Driscoll, Pettit et al., 2017; Marks and Goard, 2021; Deitch, Rubin and Ziv, 2021; Driscoll, Duncker and Harvey, 2022). The combination of these two seemingly contradictory developments are an automatic byproduct of a CML: Its cognitive map continuously shifts when new parts of the environment or new

actions are explored, also in parts of the cognitive map that model previously explored parts of the environment, see movie[6]. The reason is that the CML continuously strives to optimize the geometry of its cognitive map in order to reduce prediction errors. But the later refinements of the cognitive map tend to have little impact on its planning and problem solving capability, since only the angles between the representations of actions and state differences are salient for its planning capability, not their actual neural representations.

Learning to predict future observations, which is the heart of the CML, has already frequently been proposed as a key principle of brain function (R. P. Rao and Ballard, 1999; Friston, 2018; Keller and Mrsic-Flogel, 2018). However, work on predictive coding in the brain has so far focused on the creation of internal models for the environment, rather than on its capability to support planning and problem solving. Hence the CML opens the door to a new application domain for predictive coding.

The CML shares some features with Transformers, such as an emphasis on learning relations between observations, its reliance on self-supervised learning of predictions, and its creation of malleable high-D internal states. But its application domain is complementary to that of Transformers: It does not just passively observe and predict, but is designed to support an active agent whose actions impact its future sensory inputs. An important difference from the learning perspective is that the CML does not require BPTT or BP, local Hebbian learning suffices. In addition, it can learn from relatively small datasets. In future work, one could combine standard Transformers with a CML in order to enhance performance in tasks where both active action selection and predictions from longer sequences of observation are desired. Other interesting extension of CMLs are hierarchical CMLs that can simultaneously plan on several levels of abstraction, see (McDonald, 2023) for a first step in this direction.

Models for learning goal directed action sequences have so far been primarily based on the conceptual framework of Reinforcement Learning (RL). But this approach makes it difficult to understand how the brain attains the capability to respond to new challenges in a very flexible manner, since the results of RL tends to be specialized for reaching a particular goal. The CML provides an interesting alternative to RL-based models that can explain the behavioral flexibility that brains achieve, often just on the basis of self-supervised learning during playful exploration.

# Methods

## Mathematical description

The observation $o_t$ is embedded into the state-space of the CML using the embedding matrix $Q$:

$$s_t = Q \cdot o_t. \tag{10}$$

This also holds for the target state: $s^* = Qo^*$.

The next state prediction of the CML can be written as:

$$\hat{s}_{t+1} = s_t + Va_t. \tag{11}$$

During planning one first computes the current utility values for all actions:

$$u_t = V^T d_t, \tag{12}$$

[6]https://drive.google.com/file/d/19ziVY5PqXdbRk6jvUJECnTY7G6aKYxDN/view?usp=sharing

where $d_t$ is the vector pointing from the current state to the target state $s^* - s_t$.

To account for scenarios where not all actions are applicable in each state, or where different actions incur different costs, the vector of utility values for all actions is element-wise multiplied with the vector of affordance values $g_t$ to yield the vector of eligibilities for all actions, see equ. 6.

To select an action, the action with the highest eligibility is selected by applying the WTA function to the vector of eligibility values of all possible actions, see equ. 7.

## Details to planning on abstract graphs

For planning on an abstract graph every node is encoded by a unique one-hot encoded observation $o_t$. Every action $a_t$ corresponds to traversing an edge of the graph in a certain direction. Hence, there are two actions for every edge of the graph, which are all encoded by one-hot vectors. To explore the graphs, the CMLs were allowed to traverse the graph, taking 200 random trajectories through the environment where each trajectory was of length 32 steps.

As a baseline comparison, the Dijkstra graph search algorithm (see Alg. 1) was employed. It is noteworthy that in this task only graphs with equally weighted edges are considered, and therefore $A^*$ search is equivalent to Dijkstra, as there are also no heuristics that can be applied to the type of graph that we consider.

The initial values for $Q$ and $V$ were drawn from a Gaussian distribution with $\mu = 0$ and $\sigma = 0.1$. The dimensionality of the state space of the CML was 200 on the random graph, 390 on the graph with multiple paths of the same length, 310 on the graph with dead ends and 390 on the small world graph. The learning rates were $\eta_q = 0.014$ and $\eta_v = 0.004$ for the random graph, $\eta_q = 0.018$ and $\eta_v = 0.0025$ for the small world graph, $\eta_q = 0.014$ and $\eta_v = 0.002$ for the graph with multiple paths of the same length, c$\eta_q = 0.011$ and $\eta_v = 0.001$ for the graph with dead ends.

In all tasks the function $f_a$ was chosen to be a Winner-Take-All (WTA) function, which takes a vector as an argument and returns a vector indicating the position of the highest valued element of the input vector with a 1 and setting all other elements to 0, see equ. 13.

$$\mathrm{WTA}(x) = v, \quad \text{where } v_i = \begin{cases} 1 & x_i = \max(x) \\ 0 & \text{else.} \end{cases} \tag{13}$$

Note, in equ. 13 if $\exists\ i,\ j,\ i \neq j: \quad v_i = v_j = \max(x)$, then the WTA function would return a one-hot encoded vector indicating index with the lowest value.

The vector $g_t \in \mathbb{R}^{n_a}$ of affordance values is defined for the case of unweighted graphs as a k-hot encoded vector, whose components with 1 indicate that the corresponding action can be executed in the current state (observation), i.e., corresponds to an outgoing edge from the current node in the graph.

### Details to weighted graphs

The affordance values for weighted graphs were computed by dividing 1 by the weight of the corresponding edge in the graph. This way, selecting actions with large weights (costs) is discouraged. To avoid that the CML gets stuck in a loop it turned out to be useful to disallow selection of the same action twice within a trial.

## Details to 2D navigation

Each observation $o_t$ was one-hot encoded. Two different environment geometries were considered in this task: rectangular and hexagonal environments. An action was interpreted in the environment as a move from one cell in the grid to a neighbouring cell. Hence there were four actions in the rectangular environment while there were six actions in the hexagonal environment. In the environments of the 2D navigation tasks all actions were executable (affordable) in every state, as long as the action would not cause the move outside of the boundary of the grid. Consequently the affordance values in $g_t$ had value 1 for all actions, unless the action would result in moving outside of the grid, in which case it had value 0. The initial values for $Q$ were set to zeros, while the initial values for $V$ were drawn from a Gaussian distribution, with $\mu = 0$ and $\sigma = 1$. The dimensionality of the state space of the CML was 80 for the rectangular grid environment and 100 for the hexagonal grid environment. The learning rate was $\eta_q = 0.37$ and $\eta_v = 0.016$ for the rectangular grid environment and $\eta_q = 0.059$ and $\eta_v = 0.015$ for the hexagonal grid environment.

The parallelism score was obtained by computing the cosign similarity between the state differences of the observations indicated in Fig. 5b. The state difference between two observations $o^a$ and $o^b$ can be computed by simply embedding the observations into state space and then compute the difference: $Qo^b - Qo^a$.

## Details to controlling a quadruped

### Details to relating action commands $a_t$ to actions in the environment

An action remapping scheme is used to remap a one-hot encoded action resulting from the WTA to a dense action consisting of 8 torque values that can be used to control the ant model. For this mapping, only torques of strength $\pm 0.125$ were considered. As there were 8 controllable joints, this yields a set of $2^8 = 256$ different combinations of considered torques. Each of these combinations was considered as one possible action. These combinations were enumerated in a list and the one-hot encoded action $a_t$ was used as an index for selecting a combination of torques in this list. Furthermore, each action was applied 10 times in the environment to guarantee a larger change in the environment.

The initial values for the two matrices $Q$ and $V$ were drawn from a Gaussian distribution, with $\mu = 0$ and $\sigma = 0.1$, where for $V$ $\sigma = 1$ was used. The dimensionality of the state-space was of the CML 120. The learning rates were set to $\eta_q = 0.01$ and $\eta_v = 0.004$

The affordance value $g_t \in \mathbb{R}^{n_a}$ is defined as a vector, where the value of the elements was 1, unless the corresponding action would cause the model to move outside of the grid. In this case, the corresponding value in $g_t$ would be set to 0.

### Details to the observations of the ant

A detailed table explaining the 29 dimensional vector of the observation $o_t$ can be found in Tab. 1.

| index | Description |
|---|---|
| 1 | x-coordinate of the torso |
| 2 | y-coordinate of the torso |
| 3 | z-coordinate of the torso |
| 4 | x-orientation of the torso (quaternions) |
| 5 | y-orientation of the torso (quaternions) |
| 6 | z-orientation of the torso (quaternions) |
| 7 | w-orientation of the torso (quaternions) |
| 8 | angle between torso and first link on front left |
| 9 | angle between the two links on the front left |
| 10 | angle between torso and first link on front right |
| 11 | angle between the two links on the front right |
| 12 | angle between torso and first link on back left |
| 13 | angle between the two links on the back left |
| 14 | angle between torso and first link on back right |
| 15 | angle between the two links on the back right |
| 16 | x-coordinate velocity of the torso |
| 17 | y-coordinate velocity of the torso |
| 18 | z-coordinate velocity of the torso |
| 19 | x-coordinate angular velocity of the torso |
| 20 | y-coordinate angular velocity of the torso |
| 21 | z-coordinate angular velocity of the torso |
| 22 | angular velocity of angle between torso and front left link |
| 23 | angular velocity of the angle between front left links |
| 24 | angular velocity of angle between torso and front right link |
| 25 | angular velocity of the angle between front right links |
| 26 | angular velocity of angle between torso and back left link |
| 27 | angular velocity of the angle between back left links |
| 28 | angular velocity of angle between torso and back right link |
| 29 | angular velocity of the angle between back right links |

Table 1: Components of an observation $o_t$ for the case of the ant

**Details to the use of target observations for the ant**

A key feature of the CML is that every type of task is formulated as a navigation problem to a target state $s^*$ in state space, where $s^*$ results from embedding a desired target observation $o^*$ using $Q$. Consequently, it is possible to design a desired target observation $o^*$, which can be passed to the CML, which then in turn tries to find a sequence of actions with the intent of receiving a current observation $o_t$ from the environment which matches the target observation $o^*$. The resulting flexibility is underlined by the three different types of problems presented in ant controller: moving to a target location, fleeing from an adversary and chasing a target. For each of these tasks, different target observations $o^*$ are computed and passed to the CML. All three tasks are based on navigation and therefore use the first two fields in the observation (see Tab. 1) to set a desired target location using Cartesian coordinates. In the moving to target location task, the target location is chosen to be fixed and has an initial distance of 20 meters away from the ant. In the chasing a target task, the target location is moving and is therefore updated every time step. In

the fleeing from an adversary task, the target location is updated every time step as well and set to 5 meters away from the ant, in the direction pointing directly away from the adversary. Furthermore, the target location is passed to the ant relative to itself and not the absolute location as defined by the global coordinate system. To transform the absolute location to a target location relative to the ant first a vector $v_{at}$ which points from the ant to the target location is computed. This vector can be written in polar coordinates, where $|v_{at}|$ is the magnitude and $\phi_{at}$ corresponds to the angle. To correct for the error that would occur if the ant itself is rotated, we deduct the angle of the ant itself (to the x-axis) $\phi_a$ from $\phi_{at}$. Finally, the target coordinates can be computed by adding the polar vector with the angle $\phi_{at} - \phi_a$ and magnitude $|v_{at}|$ to the current position of the ant. As can be seen in Tab. 1 the observation also consists of 27 other fields in addition to the coordinates. The remaining fields of $o_t^*$ are chosen to be equal to $o^t$, as this induces the CML to only change the current position of the center of its torso, and not any other variable from $o_t$.

**Details to the regularization of $Q$**

During the learning process the CML might try to focus on predicting some parts of the next observation $o_t$ which are easily predictable in order to minimize the prediction error. An example for components of the observation $o_t$ that are easy to predict are the angles of the joints, while the spatial position of the ant is harder to predict. A problem can therefore arise when the CML adapts the embedding $Q$ so that the spatial position has little impact on the vector in the state space onto which the observation vector is mapped. To ensure that variables that are important for planning are well-represented in the state space, the embedding matrix $Q$ is regularized such that every column has a minimum norm of 1. Furthermore, $Q^T$ was also regularized to reconstruct the observation $o_t$ in this task using a similar learning rule as for all other optimization processes. This also induces the CML to work with state representations that are very informative about the observations to which they correspond.

**Details to the computation of $g_t$**

Actions can entail one of two possible local actions per joint, which apply torques in one of the two directions in which the joint can rotate. Furthermore, every joint has two limits, which represent the minimum and maximum angle that the joint can have. The affordance of an action depends for the ant controller on the current angle of each joint. If a joint is already close to one of its limits, the action bringing it even closer to the limit receives a low affordance value (close to 0) and the action moving it away from the limit receives a high affordance value (close to 1). As the CML considers the actions to be one-hot encoded, the affordance values $g_t$ are computed by averaging over the individual affordance values that result from this heuristics for each joint.

**Details to the sampling of actions**

To allow the CML to explore the environment, actions are sampled randomly. The sampling process takes the bounds of the angles into account, selecting torque values which are unlikely to move the joint too close to the bound with a higher probability.

### Details to the optimization process

The loss function used for the optimization of $Q$ and $V$ is the mean squared error of the next state prediction error and can be found in equ. 14:

$$L = \frac{1}{2} \sum_{j}^{n_s} (s_t[j] - \hat{s}_t[j])^2, \tag{14}$$

where $n_s$ is the dimensionality of the state space. The delta rule, which is used to optimize the weights $Q$ and $V$, has been shown to be strongly related to this loss function, as it can be derived from gradient descent for a single layer neural network (Sutton, 1992).

To decrease the number of required trajectories in the environment, experienced trajectories were stored in an episodic memory replay buffer and reused during the learning mode.

Furthermore, the weight updates as described in equ. 2, and 3 were accumulated over a single trajectory and applied at the end of the trajectory.

### Acknowledgements

We would like to thank Christopher Summerfield, Guozhang Chen, Yujie Wu, and Yukun Yang for helpful comments on an earlier version of this paper. This research was partially supported by the Human Brain Project (Grant Agreement number 785907) of the European Union and a grant from Intel.

## References

Behrens, Timothy EJ et al. (2018). 'What is a cognitive map? Organizing knowledge for flexible behavior'. In: *Neuron* 100.2, pp. 490–509.

Brockman, Greg et al. (2016). 'Openai gym'. In: *arXiv preprint arXiv:1606.01540*.

Buzsáki, György (2019). *The brain from inside out*. Oxford University Press.

Cooper, Leon N and Mark F Bear (2012). 'The BCM theory of synapse modification at 30: interaction of theory with experiment'. In: *Nature Reviews Neuroscience* 13.11, pp. 798–810.

Davies, Mike et al. (2021). 'Advancing neuromorphic computing with loihi: A survey of results and outlook'. In: *Proceedings of the IEEE* 109.5, pp. 911–934.

Deitch, Daniel, Alon Rubin and Yaniv Ziv (2021). 'Representational drift in the mouse visual cortex'. In: *Current biology* 31.19, pp. 4327–4339.

Diba, Kamran and György Buzsáki (2007). 'Forward and reverse hippocampal place-cell sequences during ripples'. In: *Nature neuroscience* 10.10, pp. 1241–1242.

Driscoll, Laura N, Lea Duncker and Christopher D Harvey (2022). 'Representational drift: Emerging theories for continual learning and experimental future directions'. In: *Current Opinion in Neurobiology* 76, p. 102609.

Driscoll, Laura N, Noah L Pettit et al. (2017). 'Dynamic reorganization of neuronal activity patterns in parietal cortex'. In: *Cell* 170.5, pp. 986–999.

Friston, Karl (2018). 'Does predictive coding have a future?' In: *Nature neuroscience* 21.8, pp. 1019–1021.

Green, Laura et al. (2022). 'Action-driven remapping of hippocampal neuronal populations in jumping rats'. In: *Proceedings of the National Academy of Sciences* 119.26, e2122141119.

Green, Samantha J et al. (2020). 'Chimpanzees use least-cost routes to out-of-sight goals'. In: *Current Biology* 30.22, pp. 4528–4533.

Gupta, Anoopum S et al. (2010). 'Hippocampal replay is not a simple function of experience'. In: *Neuron* 65.5, pp. 695–705.

Hertz, John, Anders Krogh and Richard G Palmer (2018). *Introduction to the theory of neural computation*. CRC Press.

Ito, Takuya et al. (2022). 'Compositional generalization through abstract representations in human and artificial neural networks'. In: *Advances in Neural Information Processing Systems* 35, pp. 32225–32239.

Izhikevich, Eugene M and Niraj S Desai (2003). 'Relating stdp to bcm'. In: *Neural computation* 15.7, pp. 1511–1523.

Keller, Georg B and Thomas D Mrsic-Flogel (2018). 'Predictive processing: a canonical cortical computation'. In: *Neuron* 100.2, pp. 424–435.

Khan, Muhammad Mukaram et al. (2008). 'SpiNNaker: mapping neural networks onto a massively-parallel chip multiprocessor'. In: *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*. Ieee, pp. 2849–2856.

Kleyko, Denis, Mike Davies et al. (2022). 'Vector symbolic architectures as a computing framework for emerging hardware'. In: *Proceedings of the IEEE* 110.10, pp. 1538–1571.

Kleyko, Denis, Dmitri Rachkovskij et al. (2023). 'A survey on hyperdimensional computing aka vector symbolic architectures, part ii: Applications, cognitive models, and challenges'. In: *ACM Computing Surveys* 55.9, pp. 1–52.

Kohonen, Teuvo (1990). 'The self-organizing map'. In: *Proceedings of the IEEE* 78.9, pp. 1464–1480.

Krotov, Dmitry and John Hopfield (2020). 'Large associative memory problem in neurobiology and machine learning'. In: *arXiv preprint arXiv:2008.06996*.

Maass, Wolfgang (2011). 'Liquid state machines: motivation, theory, and applications'. In: *Computability in context: computation and logic in the real world*, pp. 275–296.

Marks, Tyler D and Michael J Goard (2021). 'Stimulus-dependent representational drift in primary visual cortex'. In: *Nature communications* 12.1, p. 5169.

Mattar, Marcelo G and Máté Lengyel (2022). 'Planning in the brain'. In: *Neuron*.

McDonald, Nathan (2023). 'Modularizing and Assembling Cognitive Map Learners via Hyperdimensional Computing'. In: *arXiv preprint arXiv:2304.04734*.

Moser, Edvard I, Emilio Kropff, May-Britt Moser et al. (2008). 'Place cells, grid cells, and the brain's spatial representation system'. In: *Annual review of neuroscience* 31.1, pp. 69–89.

Pezzulo, Giovanni et al. (2007). 'From actions to goals and vice-versa: Theoretical analysis and models of the ideomotor principle and tote'. In: *Anticipatory Behavior in Adaptive Learning Systems: From Brains to Individual and Social Behavior 3*. Springer, pp. 73–93.

Pfeiffer, Brad E and David J Foster (2013). 'Hippocampal place-cell sequences depict future paths to remembered goals'. In: *Nature* 497.7447, pp. 74–79.

Piray, Payam and Nathaniel D Daw (2021). 'Linear reinforcement learning in planning, grid fields, and cognitive control'. In: *Nature communications* 12.1, p. 4942.

Plate, Tony A (1995). 'Holographic reduced representations'. In: *IEEE Transactions on Neural networks* 6.3, pp. 623–641.

Ramsauer, Hubert et al. (2020). 'Hopfield networks is all you need'. In: *arXiv preprint arXiv:2008.02217*.

Rao, Mingyi et al. (2023). 'Thousands of conductance levels in memristors integrated on CMOS'. In: *Nature* 615.7954, pp. 823–829.

Rao, Rajesh PN and Dana H Ballard (1999). 'Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects'. In: *Nature neuroscience* 2.1, pp. 79–87.

Ritter, Sam et al. (2020). 'Rapid task-solving in novel environments'. In: *arXiv preprint arXiv:2006.03662*.

Russell, Stuart and Peter Norvig (2020). 'Artificial intelligence: A modern approach, 4th Edition'. In: *Pretence artificial Hall series in intelligence*.

Shelley, Laura E and Douglas A Nitz (2021). 'Locomotor action sequences impact the scale of representation in hippocampus and posterior parietal cortex'. In: *Hippocampus* 31.7, pp. 677–689.

Strukov, Dmitri B et al. (2008). 'The missing memristor found'. In: *nature* 453.7191, pp. 80–83.

Summerfield, Christopher (2022). *Natural General Intelligence: How understanding the brain can help us build AI*. Oxford University Press.

Sutton, Richard S (1992). 'Adapting bias by gradient descent: An incremental version of delta-bar-delta'. In: *AAAI*. Citeseer, pp. 171–176.

Tanaka, Gouhei et al. (2019). 'Recent advances in physical reservoir computing: A review'. In: *Neural Networks* 115, pp. 100–123.

Todorov, Emanuel, Tom Erez and Yuval Tassa (2012). 'Mujoco: A physics engine for model-based control'. In: *2012 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, pp. 5026–5033.

Tolman, Edward C (1948). 'Cognitive maps in rats and men.' In: *Psychological review* 55.4, p. 189.

Toyoizumi, Taro et al. (2005). 'Generalized Bienenstock–Cooper–Munro rule for spiking neurons that maximizes information transmission'. In: *Proceedings of the National Academy of Sciences* 102.14, pp. 5239–5244.

Vaswani, Ashish et al. (2017). 'Attention is all you need'. In: *Advances in neural information processing systems*, pp. 5998–6008.

Verma, Naveen et al. (2019). 'In-memory computing: Advances and prospects'. In: *IEEE Solid-State Circuits Magazine* 11.3, pp. 43–55.

Whittington, James CR, David McCaffary et al. (2022). 'How to build a cognitive map: insights from models of the hippocampal formation'. In: *arXiv preprint arXiv:2202.01682*.

Whittington, James CR, Timothy H Muller et al. (2020). 'The Tolman-Eichenbaum machine: Unifying space and relational memory through generalization in the hippocampal formation'. In: *Cell* 183.5, pp. 1249–1263.

Whittington, James CR, Joseph Warren and Timothy EJ Behrens (2021). 'Relating transformers to models and neural representations of the hippocampal formation'. In: *arXiv preprint arXiv:2112.04035*.