Local prediction-learning enables neural networks to plan

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

17th October 2022

Abstract

The capability to plan a sequence of actions toward a given goal is a cornerstone of higher cognitive function. But compelling models for planning in the brain, or more generally in any type of neural network, are missing. We present a simple model for planning in neural networks, the Cognitive Map Learner (CML), that can achieve high performance on a variety of tasks by learning a cognitive map of the environment. The way how the CML constructs a cognitive map is based on a fundamental insight from neuroscience: Observations from the environment acquire meaning for the organism primarily through performing actions that change them. The CML also provides a viable alternative to reinforcement learning in robotics, since it learns faster and becomes more flexible, due to its task-agnostic design principles. The design of the CML is also of interest from the perspective of the relationship between self-attention networks (Transformers) and neural networks, since it combines attractive features of both.

Introduction

Planning capability is a cornerstone of higher brain function. But it remains one of the most elusive cognitive processes at the neural implementation level (Mattar and Lengyel, 2022). Optimal planning algorithms have been developed in AI, such as Dijkstra's algorithm (see Methods Alg. 1), which allows finding a trajectory of shortest distance in a graph. An extension to Dijkstra's algorithm is the A* algorithm, which can improve the speed at which the goal is found by using a heuristic that prioritizes visiting nodes that are closer to the goal. The A* implementation requires data structures and algorithms with nested loops that can be efficiently implemented in a digital computer, but not in a neural network. Furthermore, A* requires complete knowledge of the planning space a-priori, it provides no method for learning it, e.g., by using experienced trajectories. Both factors make it unlikely that the brain employs a similar method for planning. A related open problem from the perspective of neuroscience and cognitive science is the nature of internal representations of states in the planning space which the brain employs. Cognitive maps have emerged as the leading metaphor for the internal representation of these states (Tolman, 1948; Behrens et al., 2018; Whittington, McCaffary et al., 2022). These models are inspired by the results about the representations of locations in a 2D maze through specialized neural network architectures in the hippocampus and entorhinal cortex (E. I. Moser, Kropff, M.-B. Moser et al., 2008). Recent experimental data suggest that the organisation of these cognitive maps for navigation in a 2D environment encodes not only the spatial structure through place- and grid cells, but also specific actions that lead from one location to another (Shelley and Nitz, 2021; Green et al., 2022). Furthermore, humans are able to plan a trajectory to a goal state not only in low-dimensional spatial environments, but also in abstract high-dimensional concept spaces (Behrens et al., 2018). Some experimental evidence suggests that the brain employs some form of a cognitive map also for that. But it has remained open how these cognitive maps are learnt and used for planning in neural networks of the brain. One intriguing suggestion for the representation and organisation of state representations in the brain was made by (Buzsáki, 2019): Sensory inputs acquire meaning for the brain primarily through self-initiated actions which change them, such as making a step forward or turning off the light. Hence state representations and cognitive maps should fuse information from the environment that characterises states with information about the actions that change these states.

Based on this brain-derived principle we propose a simple model, a Cognitive Map Learner (CML), for learning and using cognitive maps. The self-supervised learning on which it is based can be implemented with simple local rules for synaptic plasticity that are consistent with experimental data. For planning domains where actions are reversible it can nevertheless plan in the cognitive maps that it has learnt from random exploration almost as well as the less flexible and neural-network-implementation-unfriendly gold standard A* that requires complete knowledge of the environment. We demonstrate the capabilities of the CML in applications to the arguably most difficult planning scenario, a non-planar random graph, as well as on a 2D navigation task that requires spatial generalization, and a third completely different task: robot control for a quadruped.

Results

Design principles for a cognitive map learner (CML) that is able to plan

Basic ideas and concepts

The core component of the CML is its high dimensional state space $\mathcal{S} \in \mathbb{R}^{n_s}$, which loosely corresponds to its cognitive map (Behrens et al., 2018), i.e., to the brain representation of the environment and/or configurations of the body. Taking actions amounts in the CML to navigation in the state space $\mathcal{S}$. In contrast to many other approaches, we do not assume that this state-space is given a-priori. Rather, it emerges through self-supervised local prediction-learning: by observing how an action changes the environment or the body. In particular, prediction-learning re-organizes the geometry of the state space so that Euclidean distance between two states reflects the number of actions that are needed to move from one to the other, rather than their similarity or spatial distance. Since the distance between two states A and B is symmetric, this intuition is only applicable if one can move from back from state B to a state A by inverting the actions of a trajectory from A to B.

State representations in the CML are shaped by two constraints:

i) The transformation that maps (embeds) sensory inputs and/or internal feedback from the body, lumped together under the term "observations" in the following, into a corresponding state is linear.

ii) The prediction of the next state that results from an action can be approximated by a linear function applied to the preceding state and a suitable internal representation of the action.

The second constraint induces two benefits for biologically oriented models and neuromorphic hardware. One is that these maps can be represented by a single layer of neurons, see the layers marked green in Fig. 1 a, b, and learnt by simple local rules for online synaptic plasticity. In particular, they do not require complex offline learning schemes such as BPTT which is needed for training transformers(Vaswani et al., 2017). Another essential benefit is that the inverse map is also linear, and hence can also be learnt by a single layer of (linear) neurons through Hebbian-like local synaptic plasticity. Having a linear inverse model also greatly simplifies planning, since the sequence of actions that is needed to reach a given goal is produced by the CML as a linear sum of these actions. Hence a simple nonlinear operation can extract from this sum that action which has the largest affordance (Khetarpal et al., 2020) in the current state, and propose it as the next action to be executed. Affordances are learnt by the CML along with predictions of the next state through self-supervised learning.

Network structure

A neural network implementation of this CML consists of two pools of linear neurons, marked green in Fig. 1 a, b. During exploration, actions can be generated by some arbitrary exploration strategy. Only linear computational operations, represented through three matrices, are applied by the CML to process these experiences, see Fig. 1 a. During planning, three simple nonlinear operations are applied, as indicated in the blue box ("planning core") of Fig. 1 b, c. Sigmoid functions are used for producing estimates for the affordance (applicability) of each possible action in the current state. The result is element-wise multiplied with the current "utility" of each possible action for the current planning goal s^*. Winner-Take-All (WTA) is applied to the resulting vector in order to determine the action that the CML applies next.

Learning and planning

CMLs have two modes of operation: learning of state predictions and affordances from autonomous explorations and using learnt information for planning action sequences (trajectories) to arbitrarily given target states s^*. Although self-supervised learning can continue during planning, for simplicity we describe here learning and planning as two separate phases:

1. In the learning mode the CML is allowed to explore autonomously, using any exploration strategy it likes, starting at its current state. This results in a sequence (o_t, a_t) of observations o_t and actions a_t for every step t of the exploration sequence. Three matrices W_q, W_k, and W_v are optimised through self-supervised learning during this exploration, see Fig. 1 a. The matrix W_q embeds observations into the state space $\mathcal{S}$. The matrix W_k maps state representations to values which estimate the affordance of each action in the current state. Most importantly, the matrix W_v maps actions to estimates of what change in the state they are likely to cause. Its learning is directly driven by prediction-errors. The inverse of this matrix provides the key for planning, but no explicit matrix inversion is needed for that: An approximation of the inverse of W_v can be learnt online during the learning mode, together with the matrix itself.

   Note that replay of exploration episodes (o_t, a_t) during learning reduces in general the required amount of exploration.

2. In the planning mode, the CML uses the learnt cognitive map for solving online planning tasks. To reach a given target state s^*, which typically results from embedding a target observation o^* into the state space with the learnt linear map W_q, the CML computes with the help of the inverse of W_v a utility score u_t for every action. This utility score provides information on how useful each of them would be for moving towards s^*. The resulting vector u_t is multiplied element-wise in a shallow nonlinear planning core, depicted as a blue box in Fig. 1 c, with a vector g_t. This vector estimates, with the help of W_k, the affordance of each action in the current state. The next action a_t is selected from the resulting vector through a Winner-Take-All (WTA) operation.

Practically, the CML can also learn during planning. This not only speeds up the process, but also has the advantage that learnt internal representations become most precise for those parts of the state space that are most frequently visited for actually arising planning tasks. A separate prior learning or exploration phase is only needed to make sure that for each state a fair number of possible actions is tried out.

We will present applications of the CML to three very different planning scenarios. For simplicity, we completed there the learning mode before activating the planning mode, using a uniform random selection of actions during learning. But besides continuing to learn during subsequent planning one can also interleave prediction-learning and planning phases, so that more sophisticated exploration strategies arise during learning that take outcomes of preceding planning phases into account.

Note that in contrast to using A* for planning a trajectory, the CML breaks down the planning of a trajectory into iterated real-time planning and execution of the next action, thereby eliminating

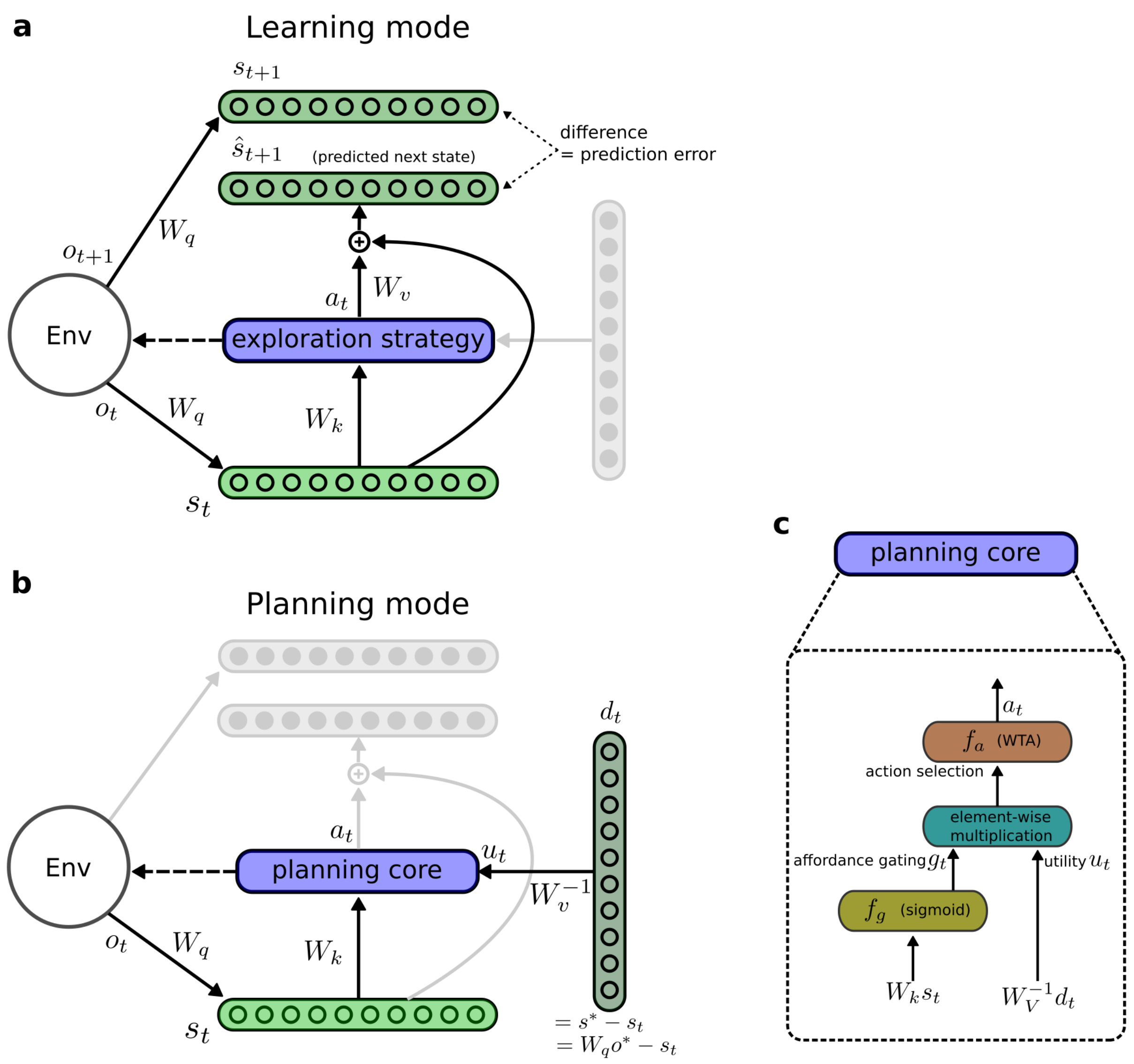

Figure 1: **Structure of the Cognitive Map Learner (CML) a** Components of the CML used in the learning mode, where local synaptic plasticity is primarily driven by prediction errors for the outcomes of actions. Internal predictions of the next state $s_t + 1$ are produced by adding to the current state s_t a learnt linear embedding of the action a_t that is applied at step t. The action selection process (marked in blue) can be implemented in the learning mode with any exploration strategy. **b** Components of the CML used in the planning mode. Here the action selection process (marked again in blue) is implemented by the planning core, which applies simple nonlinear operations such as softmax and multiplication in order to propose a next action a_t that is both feasible in the current state and moves in the direction of the target state in the learnt state space (cognitive map). **c** Detailed description of the inner workings of the planning core of the CML. Note that the learning mode can run concurrently with planning as soon as a planned action is executed.

the need to store a planned trajectory. It also makes the planning more flexible, since suddenly appearing contingencies, e.g. a roadblock, or modifications of the goal can effortlessly be taken into account. Surprisingly, the trajectories that result from this online planning approach of the CML turn out to rival in their efficiency the performance of the gold standard A*, as we will demonstrate for planning in a random graph. But in contrast to A*, for planning in 2D mazes the CML is able to take the geometry of this environment into account and propose shortcuts during planning that it never encountered during exploration. Finally, the CML can also be used for online control of complex plants, as we will show in our last demo.

Mathematical description of the CML

We describe here the previously sketched computations and learning processes of a CML with precise formulas. First, the mapping of an observation $o_t \in \mathbb{R}^{n_o}$ to a state $s_t \in \mathbb{R}^{n_s}$ is computed by the matrix $W_q \in \mathbb{R}^{(n_s,n_o)}$, where n_s is the used-defined dimension of the state space and n_o is the dimension of the observation space (which may also reflect the internal state of the body of a mobile agent in an application to motor control):

$$s_t = W_q o_t. \tag{1}$$

During the learning phase, we consider o_t to arise directly from the environment during online exploration, or also from replay of exploration strategies, which allows for the same trajectories to be used multiple times during the learning process. In the planning phase, o_t originates directly from the environment. The current state s_t can also be thought of as location in a learnt cognitive map.

Self-supervised learning mode:

The prediction of the next state $\hat{s}_{t+1} \in \mathbb{R}^{n_s}$ is computed by embedding the action into the state space with the matrix W_v and adding the result to the current state s_t:

$$\hat{s}_{t+1} = s_t + W_v a_t. \tag{2}$$

The value matrix $W_v \in \mathbb{R}^{(n_s,n_a)}$ specifies for each action how it will change the state. More precisely, W_v stores in its column i the impact that action $a_t[i]$ (ith element of the vector a_t) would have on the state. We assume that the one-hot encoded actions a_t are produced by some exploration strategy during learning. In our demos, we simply chose them uniformly randomly among all possible actions. If the chosen action could not be executed at the current state, another action was randomly chosen.

The model accounts for the fact that different actions could be available at different locations in state space. Hence, the CML computes for each potential action a_t the affordance gating value $g_t \in \mathbb{R}^{n_a}$, where n_a is the number of dimensions of the action space:

$$g_t = f_g(W_k s_t), \tag{3}$$

where f_g is a user-defined function (e.g. WTA, softmax, sigmoid). In this report f_g was always chosen to be the sigmoid function unless stated otherwise. Importantly, g_t is not directly used during in the learning mode (only in the subsequent planning mode), but it is necessary to compute it nevertheless for optimizing the matrix W_k. Note that the matrix $W_k \in \mathbb{R}^{(n_a,n_s)}$ that is used to estimate affordances has some similarity to keys in self-attention networks (Vaswani et al., 2017). Thereby the planning of the CML attends to the currently available actions. More precisely, W_k has a row for every possible action where it produces during learning an estimate of the ideal state for applying this action. If the current state is similar to this ideal state, the affordance value g_t will be close to 1, while it will be close to 0 if this action was never applied in the current state during exploration.

For self-supervised learning the CML uses the well-known delta rule (Hertz, Krogh and Palmer, 2018). This local Hebbian-like rule for synaptic plasticity is an extension of the perceptron learning rule for regression tasks. In fact, it implements stochastic gradient descent learning for a single layer of linear gates, analogous to backpropagation in multi-layer networks:

$$\Delta W_k(t) = l_k(a_t - g_t)s_t^T \tag{4}$$

$$\Delta W_v(t) = l_v(s_{t+1} - \hat{s}_{t+1})a_t^T \tag{5}$$

$$\Delta W_q(t) = l_q(\hat{s}_{t+1} - s_{t+1})o_{t+1}^T, \tag{6}$$

where l_k, l_v and l_q represent corresponding learning rates. In our demos, we accumulated for simplicity the weight updates over an entire trajectory before applying them to the weights.

Furthermore, a weight regularization scheme was applied:

$$W = \frac{W}{||W||^2}, \tag{7}$$

where the $||\cdot||^2$ operator refers to the Euclidean norm along the axis that corresponds to the state of the CML. The weights $W_k \in \mathbb{R}^{n_a,n_s}$ were restricted to have a maximum norm of 1 along the second axis, which was enforced using the update rule in equation 7 if the Euclidean norm was larger than 1. The values $W_v \in \mathbb{R}^{n_s,n_a}$ were normalized column-wise according to equation 7 after every weight update. This regularization has the important consequence that the state difference vector that is caused by an action has unit length, thereby inducing a metric in the state space that is governed by the number of actions that are needed to move from one state to another. In particular, it creates clusters of states that can be reached from each other with few actions. The matrix W_q, that transforms observations into state representations, is reminiscent of a corresponding linear operation in self-attention networks or transformers (Vaswani et al., 2017). But whereas training of these functionally extremely powerful models requires BPTT, an offline learning process whose biological plausibility is debated and which poses an obstacle for efficient implementation in neuromorphic hardware, the linear transformation W_q of a CML can be learnt through local synaptic plasticity.

Planning mode:

An essential idea of the CML is that it produces a state space (or cognitive map) whose geometry is such that the vector pointing from the current state to the target state, $d_t = s^* - s_t, \in \mathbb{R}^{n_s}$, informs the model in which direction in the state space it should move, like for planning in a Euclidean space. Surprisingly, this strategy also works for planning in generic random graphs, where the planning space has no obvious relation to the geometry of a Euclidean space. The utility scores u_t indicate to what extent each action would move the current state s_t into the direction of the target state s^*. It can be simply computed by:

$$u_t = W_v^{-1} d_t \tag{8}$$

The next action a_t is computed by combining these utility values u_t with the affordance values g_t:

$$a_t = f_a(g_t \odot u_t) \quad \in \mathbb{R}^{n_a}, \tag{9}$$

where f_a is a non-linear function (e.g. WTA, softmax) and $\odot$ corresponds to the Hadamard product (element-wise multiplication), and g_t is computed using Equ. 3. In this report, we always use WTA as this selection function f_a, yielding a one-hot encoded action. Note that it is commonly assumed that WTA-like selection functions are computed with the help of lateral inhibition in generic motifs of cortical microcircuits, see e.g. (Jonke et al., 2017).

Evaluation of the CML for planning on abstract graphs

The first demo considers a planning task on abstract graphs, consisting of nodes and undirected edges. One can view this graph as a representation of some arbitrary problem solving domain with reversible actions that may bear no relation to navigation in a physical space. We use the simplest approach where a learning phase is followed by a planning phase. In the learning phase, the CML takes 200 random walks of length 32. In this task an action corresponds to traversing an edge in a specific direction, hence there are two actions for every edge.

In the planning phase, the model is placed on a random starting node and is given an arbitrary target node. The goal is to use the previously constructed cognitive map of the abstract graph to find a short path from the start node to the target node. For the offline version of this task, where the complete graph is known from the beginning, one can solve this task also by applying the Dijkstra graph search algorithm (see Methods), or the A* search algorithm that is often described as a gold standard for problem-solving in AI, but also for planning in the brain Mattar and Lengyel, 2022. A characteristic feature of these two algorithms is that they can only be used for offline planning, while CMLs excel at online planning (see Mattar and Lengyel, 2022 for details). Online planning involves the challenge to pick an action with a limited time budget and limited information about the graph, so that exhaustive search is not an option. A CML selects an action within constant time, i.e., in real-time, no matter how large the graph is and how far the target node is away. In contrast, Dijkstra's algorithm and A* need to first compute a complete trajectory to the target node before they can produce the first action. Furthermore, A* has to be run again from scratch if the start or goal node change. Dijkstra's Algorithm has to do that also if the start node changes. In contrast, the CML can apply its learnt weights for online planning from any start to any goal state. For this task, we consider four types of abstract graphs, each posing different challenges.

Random graphs

The first category of graphs is randomly connected undirected graphs consisting of 32 nodes, where every node has between two and five edges to other nodes. One instance of this family of graphs is depicted in Fig. 3. The CML was able to plan after learning for this graph a trajectory from an arbitrary start node to an arbitrarily given target node using an average of **3.515** steps (std. deviation 1.425). This is very close to the average shortest possible path of 3.464 (std. deviation 1.352), which can be found using the Dijkstra algorithm.

While the plotting style chosen might give the intuition that these graphs are planar, especially the graph in Fig. 5, there is no spatial aspect to these graphs at all. To make this point even clearer, Fig. 2 depicts the same random graphs twice using two different graph embedding techniques.

Fig. 3 sheds some light onto the inner workings of the CML, where in the first column 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 in Fig. 3 directed edges pointing toward the goal state have a higher utility, especially when they are between the current node and the target node. The affordance values in the second column indicate which actions are available to the CML in the current state. As can be seen in Equ. 9, the action selected by the CML is related to the product of the utilities and the affordance gating variables, which conceptually is a way of selecting the best available action.

Another search task on the same graph can be found in Fig. 4, where only the utilities have been plotted.

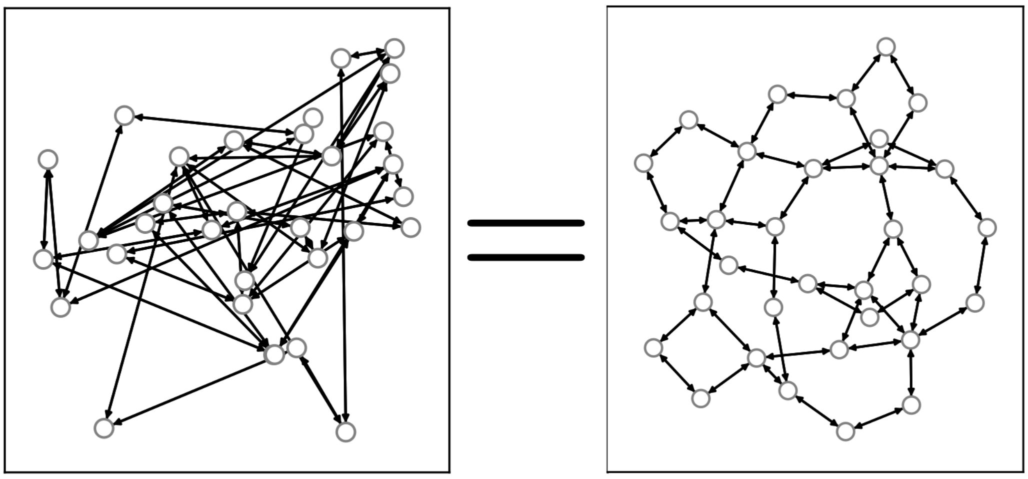

Figure 2: Two different styles of plotting a randomly connected abstract graph. On the left the nodes have been arranged randomly and on the right a force-directed graph embedding scheme (Kamada, Kawai et al., 1989) has been applied to make it look quasi-planar, although it is not planar.

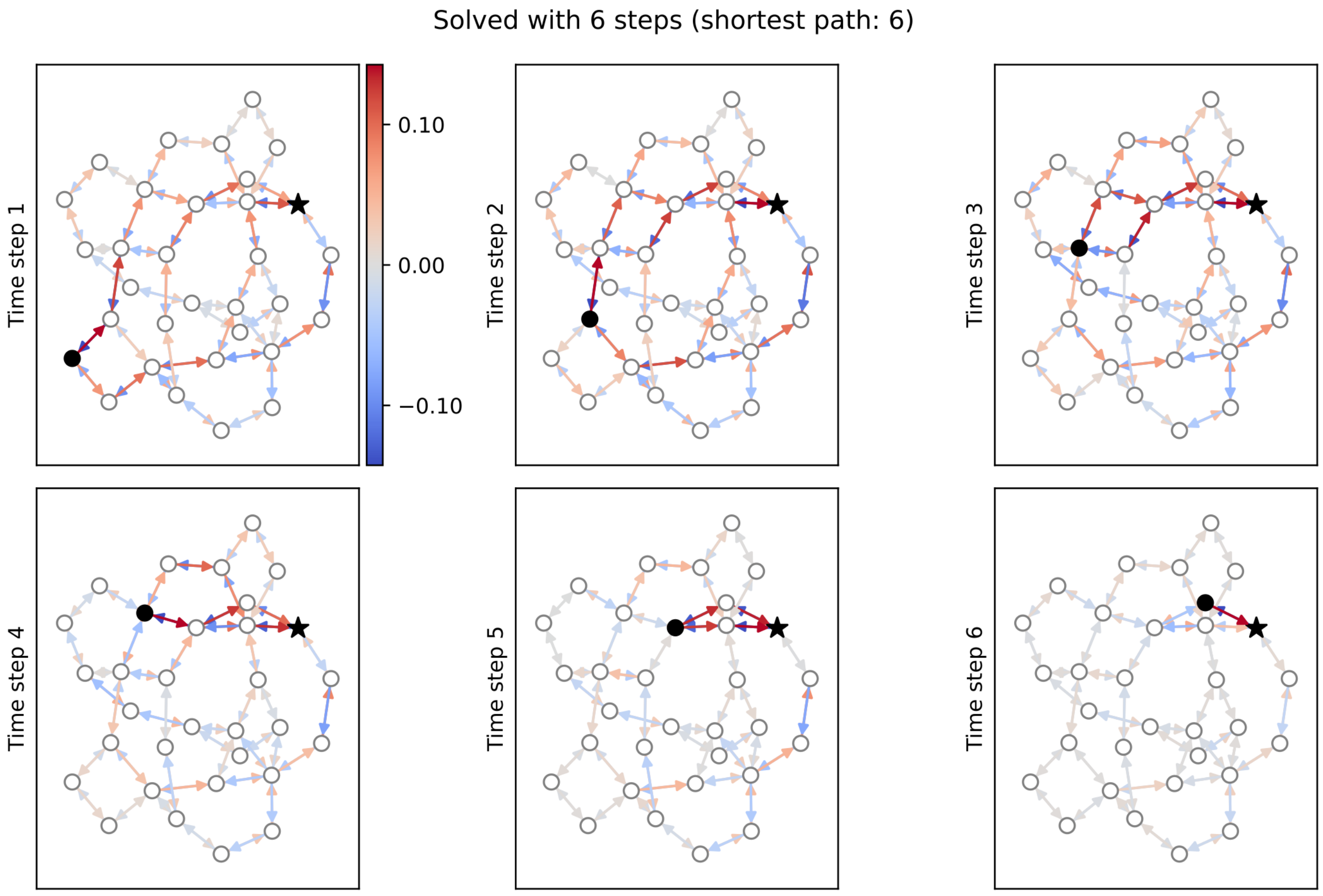

Figure 4: Another planning task on the same graph, where only the utilities have been plotted for 6 different time steps. Note, how all directed edges pointing towards the target node tend to be of high utility, especially when they are between the current node and the target node, while edges pointing away from the target node tend to have a low utility value.

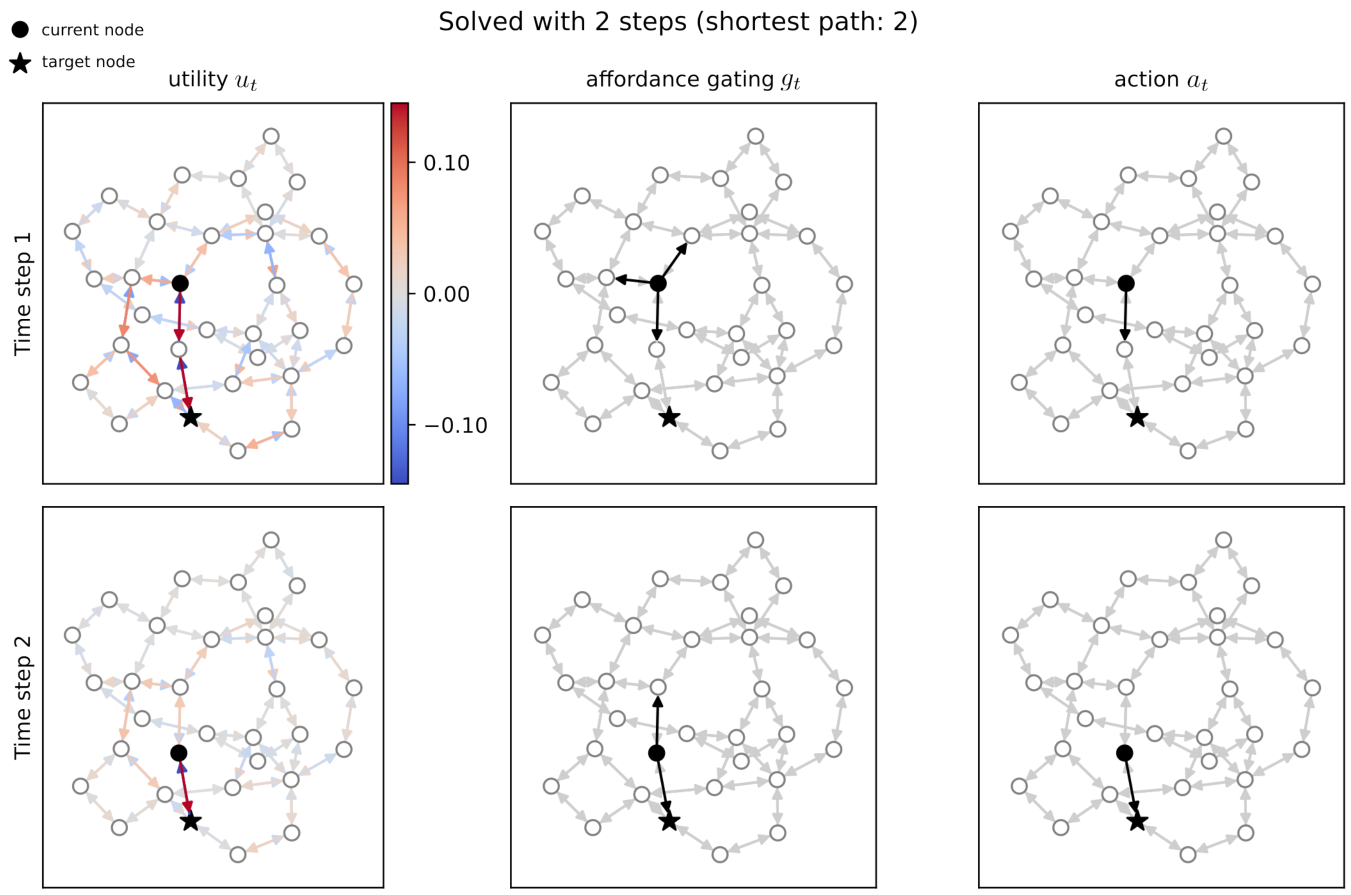

Figure 3: Inner workings of the CML while solving a graph traversal problem. The two rows correspond to two time steps, while the columns illustrate the values of different variables used in the CML. The first column indicates the utilities u_t, which indicate how useful the CML considers at the current time step traversing a given edge would be for moving towards the target node. The second column shows the affordance gating variable g_t, which informs the CML about the availability of certain actions in the current state and the third column depicts the chosen action, which is obtained by combining the utilities u_t with the affordance gating g_t as shown in Fig. 1 c.

Graphs with multiple paths of the same length

One may worry that a heuristic problem solver such as the CML has problems if there exist multiple paths of the same length to a goal, since dealing with such ambiguity might require more global knowledge of the graph structure. But we found that this feature did not impede the performance of the CML for the class of graphs with multiple paths to the same goal that we tried: The CML needed on average **2.477** (std. deviation 1.105) steps to reach the goal (average over 1000 trials), which is the same as the optimum computed using Dijkstra. In other words, the model managed to find one of the many shortest paths in every single trial. A sample trajectory can be found in Fig. 5

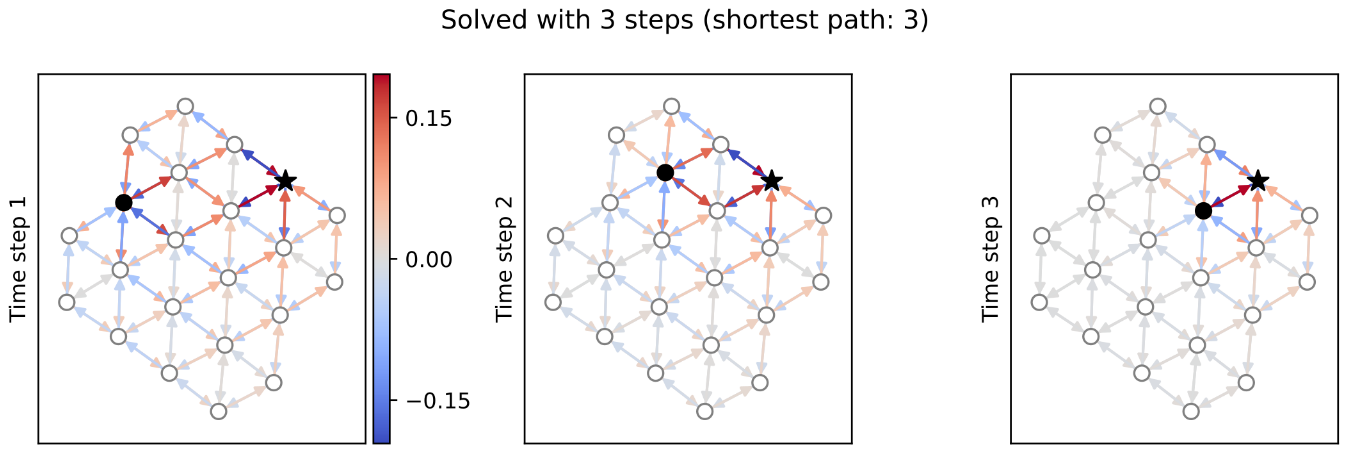

Figure 5: Utilities on a sample trial on the graph with multiple paths of the same length. One sees that mutliple options are concurrently favored by high utilities, instead of favoring only selected alternatives that might have previously been used during learning. This feature supports effective replanning in case that one of the actions becomes suddenly not available.

Graphs with dead ends

Another challenge for heuristic planners such as the CML are graphs with dead ends. These dead ends, and backing out of them, is frequently encountered during exploration, and one might suspect that such useless detours to dead ends become parts of planned trajectories to other goal states. We considered a graph consisting of 32 nodes, where half of them are dead ends. The CML was able to handle this difficulty well, reaching a goal in on average **3.133** (std. deviation 1.146) steps. This is close to the theoretical optimum of 3.121 (std. deviation 1.128) (averaged over 1000 trials). The analysis can be found in Fig. 6.

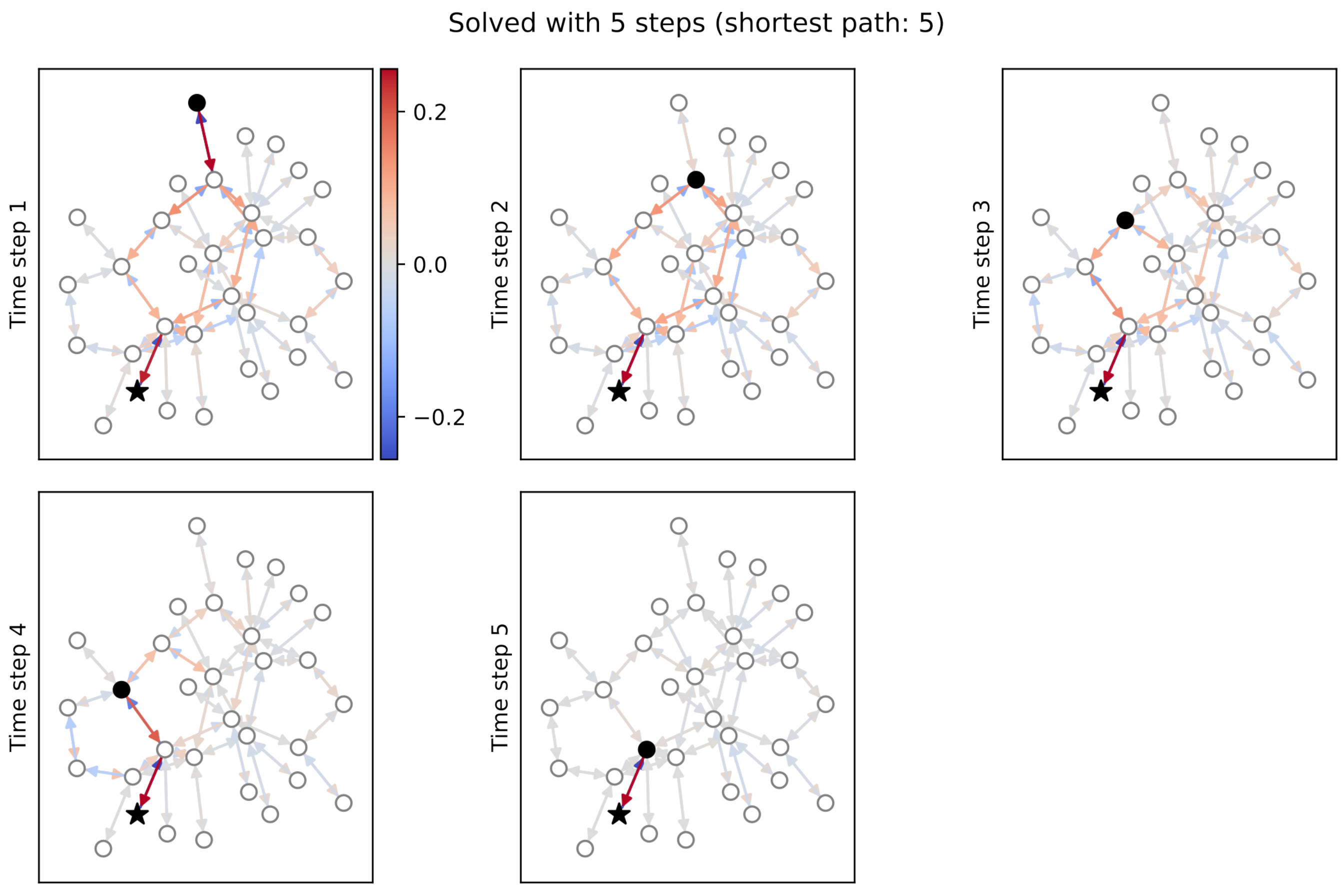

Figure 6: Utilities on a sample trial on the graph with multiple dead ends. A learning based planning method might be tempted to use segments into and out of dead ends as detours for reaching a goal state because these "detours" occurred during exploration. Note the CML does not fall into this trap, although the general uselessness of the detours was not apparent to the CML during exploration, where no target states or quality measurements were provided.

Small world graphs

Also small world graphs can be seen as challenges for heuristic planning, since it may be hard to 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. Surprisingly, even for this task the CML solved planning tasks with only **4.652** (std. deviation 2.8) steps on average (over 1000 trials), which is theoretically optimal according to the Dijkstra algorithm. In Fig. 7 only the utilities u_t have been plotted.

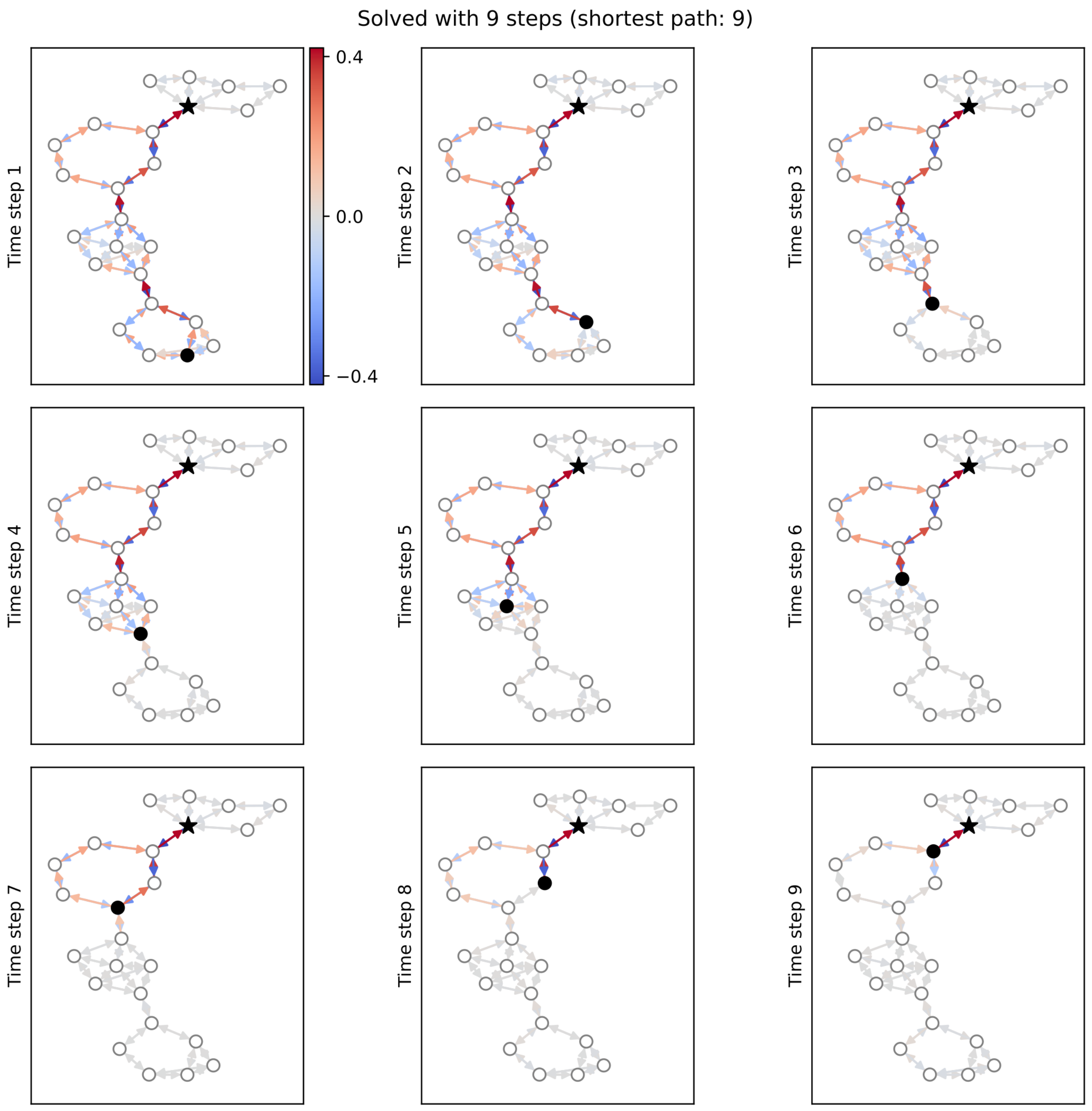

Figure 7: Utilities on a sample trial on the graph small worlds for 9 time steps of a planning process of the CML. The planned trajectory immediately moves in each cluster (small world) to that node which provides access to the next cluster.

Application of the CML for spatial navigation

Cognitive maps were first proposed to model the navigation of rodents in 2D mazes. They were found to enable a quite non-trivial generalization capability of rodents: They integrated short-cuts into their planning which they had never encountered during exploration (Gupta et al., 2010).

Therefore we tested CMLs also for this task domain. To test the spatial navigation capabilities of the CML, we designed a challenging task which requires fast learning and spatial generalization capabilities, and was inspired by the task design in Ritter et al., 2020 and Whittington, Muller et al., 2020. The environment consists of a regular grid, where on each field a certain unique sensory stimulus is located that is presented as observation to the CML through one-hot coding (, see Fig. 8b and d, where the icons correspond to the different sensory stimuli). Note that no information about the spatial location of these stimuli were given to the CML. This domain provides a nice test scenario for the capability of the CML to create a suitable cognitive map, since a perfect cognitive map (the 2D map of the maze) is known to us, but not to the CML. CML explored the environment by taking a single trajectory visiting most fields only once, see Fig. 8a, b, and c. If a chosen action was not executable during exploration, the next possible action in a fixed sequence of possible actions was chosen. Note, that the CML does not have any prior knowledge about the geometry of the environment. Instead, this property emerges naturally in the internal cognitive map during the learning process when the CML tries to make sense of observed loops, see Fig. 8a.

In the subsequent planning phase, the CML is placed on a random starting location in the environment and is tasked to navigate to a target sensory stimulus which defines a target state as indicated in Fig. 1 b, c. Fig. 8c and e show that the CML is capable of finding a path of minimal length, although it has never seen these actions being selected in these locations during training, highlighting the generalization capabilities of CMLs.

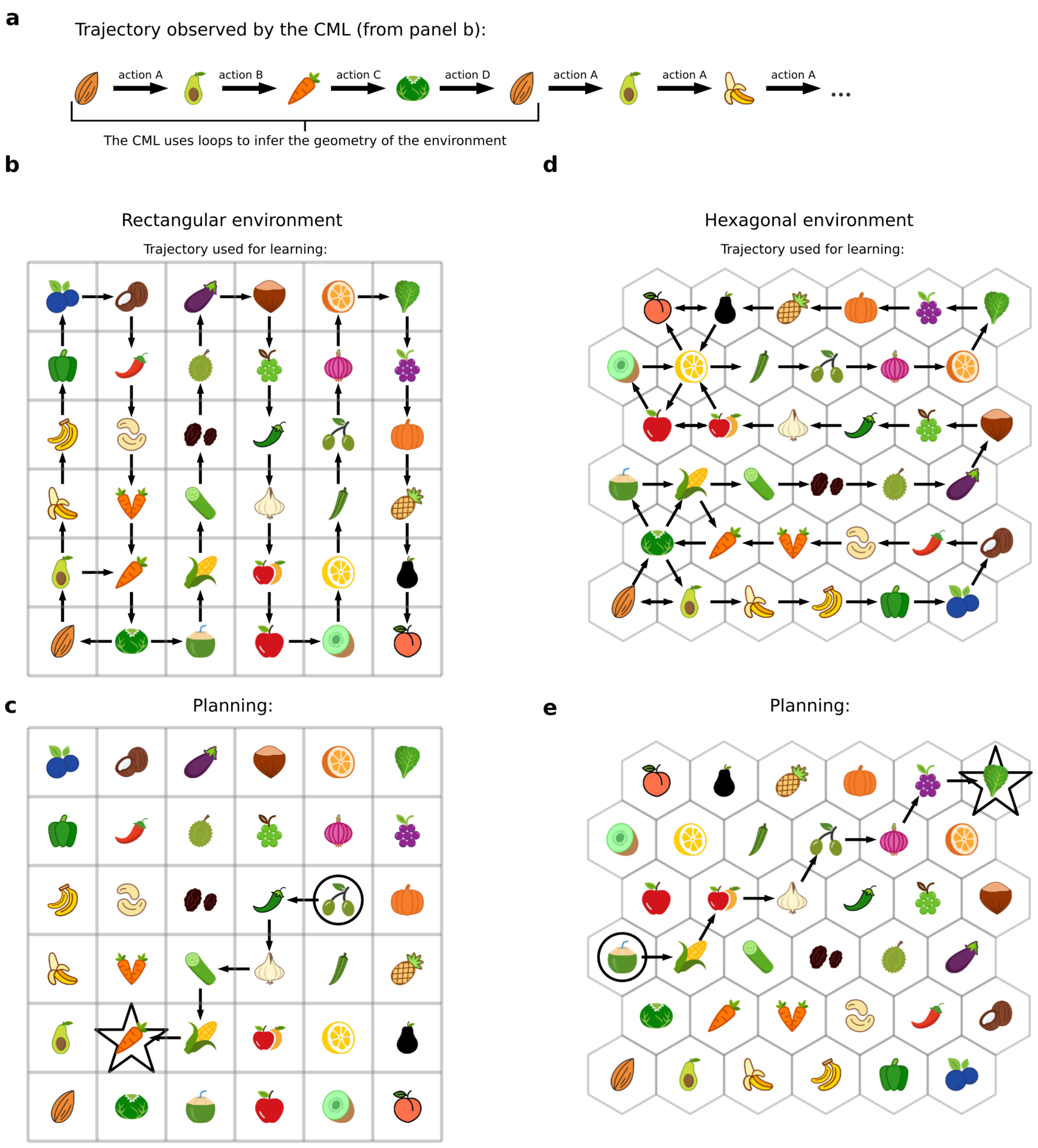
a
Trajectory observed by the CML (from panel b):
action A
action B
action C
action D
action A
action A
action A
...
The CML uses loops to infer the geometry of the environment
b
Rectangular environment
Trajectory used for learning:
d
Hexagonal environment
Trajectory used for learning:
c
Planning:
e
Planning:

Figure 8: Results on spatial navigation. Panel **a** illustrates how the CML receives as input during the learning phase a sequence of sensory stimuli as observations, which are depicted using icons, together with the corresponding sequence of actions, indicated by the arrows between the icons. During the learning process, the CML integrates such sequences into a consistent internal cognitive map, which represents the underlying 2D environment. Note, that the CML has to infer the geometry of this 2D environment based on the loops that it encounters in the trajectories of observations and actions. Panel **b** and **d** illustrate all actions that were encountered during the learning phase, in the exploration trajectories taken by the CML, which all started in the bottom left field. All icons illustrate an unique sensory stimulus (given as one-hot code) that the CML has to remember. Panel **c** and **e** show planning trials of the CML in the respective environments. Note, how the CML manages to generalize over the geometry of the environment, finding the shortest paths even if these paths require taking certain actions in certain states that were never taken during learning.

As can be seen in Fig. 8 and Fig. 9, the CML manages to construct a complete cognitive map of the environment, which enables it to find optimal trajectories for arbitrarily given start and goal locations. We applied PCA (principal component analysis) in order to she light on the organization of the geometry of the state space that emerged in the CML after exploration. Fig. 9 shows that the CML has morphed its state space during learning into a perfect 2D map of the spatial relations between the sensory stimuli.

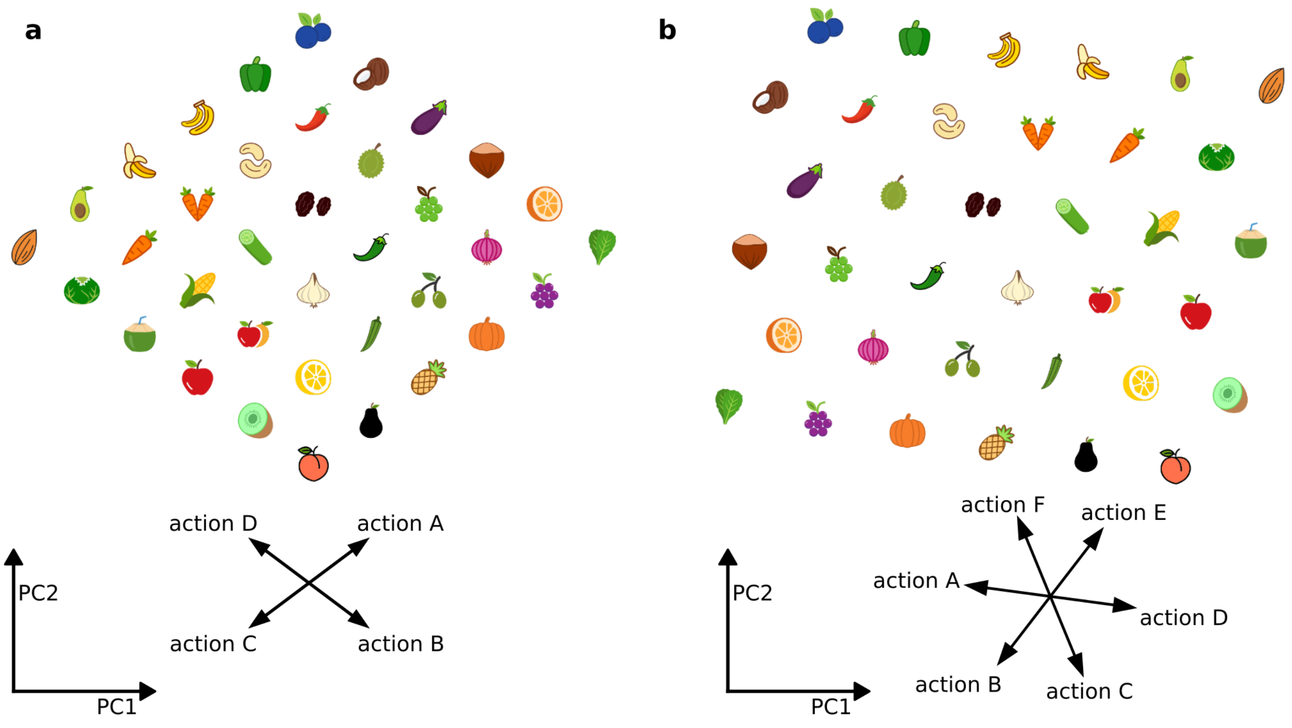

Figure 9: Principal component analysis of the state space of the CML after learning. The learnt cognitive map is visualized by performing a principal component analysis of the state space embedding of the sensory stimuli. Additionally, the same dimensionality reduction has also been applied to W_v to illustrate the movement in state space that would be caused by taking the corresponding actions. Panel **a** depicts the first two principal components for the rectangular grid and panel **b** for the hexagonal grid.

Application of the CML for learning to control locomotion of a quadruped

Our last demo shows that the CML can also be applied to a very different domain: Motor control. In fact, its planning capabilities appear to make the CML in many cases a viable alternative to traditional Reinforcement Learning (RL) approaches, which typically require large numbers of trials, and have the disadvantage that it is difficult to transfer learnt knowledge to a variation of the task. 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 these capabilities for a standard benchmark task for motor control: Locomotion of a quadruped robot (see Fig. 10a), commonly referred to as ant (in spite of the fact that ants have more than 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 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. An action of the CML consists in this case of a vector of torque values to each of the eight joints of the ant, which will be applied in the next time step of the simulated environment (1000 time steps per episode). This environment provides 29 dimensional observations o_t, which contains information about the angles of the joints, their respective speed, and about 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 video [1].

In the simplest planning challenge, the CML was tasked to navigate the ant to arbitrarily given goal locations in a distance of 10 meters away from the starting position (see video[2]). Resulting trajectories of the ant are depicted in Fig. 10b, 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.

Fig. 10c illustrates the correlation between the state prediction error and the performance of the CML in terms of the remaining distance to the target location. Importantly, the CML was never optimized to move to a goal location - it only received 200 trajectories taking random actions - but it can solve this task nevertheless, with a remaining average distance to the target of less than 2.

Due to the task agnostic nature of the CML it is possible to formulate also more complicated tasks, for example fleeing from a predator (video [3]) or chasing a target (video[4]).

Neuromorphic hardware

CMLs are of special interest for neuromorphic hardware, as they are capable of learning with local learning rules only, not requiring multi-layer backpropagation or backpropagation through time. Therefore, neuromorphic hardware platforms promise very energy-efficient implementations of CMLs, especially if the state s_t is designed in a way to promote sparseness. Note also that the 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) or other clever devices represents the learnt 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. One difficulty that can be encountered when working on neuromorphic hardware is limited precision for weights, often limited to eight bits (Davies et al., 2021). To test whether this limitation could pose a problem for CMLs, a control experiment was performed on the abstract random graph, where the bit precision was reduced to just 8 bits. We found that this did not have a large impact

[1]https://drive.google.com/file/d/1rNfcpz_s-uYnxiEBiO3Az_5DxGWpUxC8/view?usp=sharing
[2]https://drive.google.com/file/d/1bEj-phDtQ_Fy0BXZxyv9oD3VqIFfYxOd/view
[3]https://drive.google.com/file/d/1IXSTY8a1nlq7wAn1UUqasO5lnBZldBJB/view?usp=sharing
[4]https://drive.google.com/file/d/1jvnqyywef7OZYWLemiX2WILVxaiEjjy9/view?usp=sharing

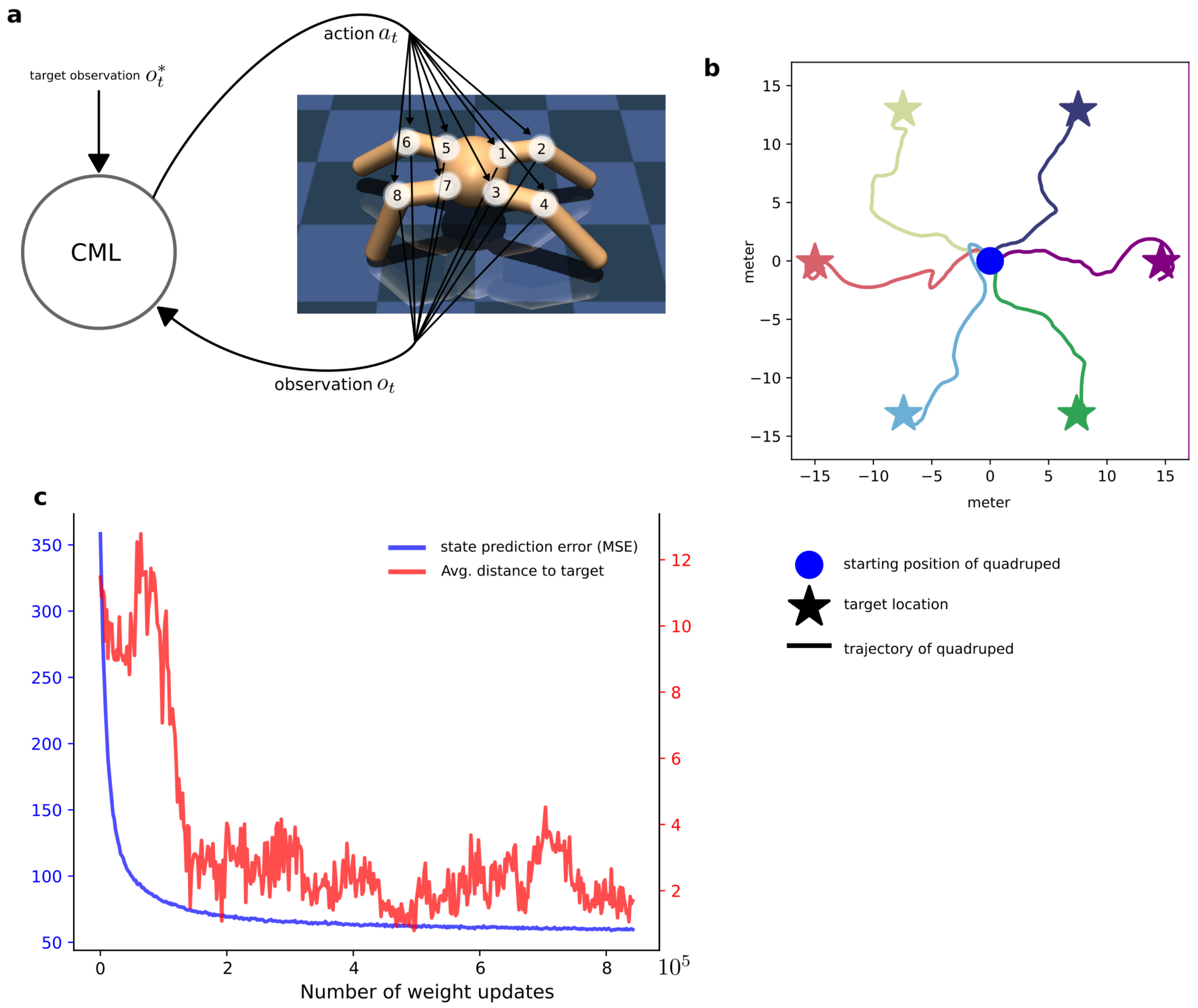

Figure 10: Application of the CML to motor control. Panel **a** depicts the quadruped ("ant"). Panel **b** illustrates the performance of the CML on this navigation task for 6 sample targets. The blue circle in the middle of the panel indicates the starting position of the ant while the stars correspond to 6 different goal states, where the corresponding filtered trajectory has been plotted in the same color. Six trajectories are shown in this panel, where the ant is 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 correlation between the next state prediction error and the average distance to the target location after the end of the episode. One sees that good planning capability sets in shortly after the prediction learning is working well.

on the performance. A degeneration in the capabilities of the CML was only detected if 6 or fewer bits were available for encoding the weights.

Related Work

As mentioned already before, 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:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V, \tag{10}$$

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. There are attempts in the literature to port the attention mechanism to neural networks, see (Ramsauer et al., 2020; Whittington, Warren and Behrens, 2021; Krotov and Hopfield, 2020). Although these network models do exhibit an approximation of the attention mechanism, other aspects of Transformers remain unreachable. For example, the neural network approximations always consider a single input token, while a Transformer uses a sequence of such tokens as input. Additionally, Transformers using self-attention compute the key, query and value matrices from the input while in the neural network counterparts they are fixed and stored in the weights of the network.

Usually, these models are based on the idea of computing a dot-product between a query q_t and a matrix containing the keys W_k. Similarly as in Equ. 10 the softmax function is applied to the result to obtain an attention vector. As the last step, this attention vector is also multiplied against a matrix containing the values. Due to the fact that these models are often viewed as variants of Hopfield networks, the matrices W_k and W_v are considered to be the same, resulting in a two layer network with symmetric connections. As an example, the model of Ramsauer et al., 2020, which is the same as the model B proposed in Krotov and Hopfield, 2020, can be written as:

$$\xi_{t+1} = \text{softmax}(\beta \xi^T X) X^T, \tag{11}$$

where ξ corresponds to the query or state s_t in CMLs and X is both the keys W_k and the values W_v, and β corresponds to a heat parameter.

Equ. 11 exhibits some similarities with CMLs. For example, the computation of the softmax can be related to Equ. 3, and the product of the softmax with the values corresponds to Equ. 2. In Equ. 2 the current state s_t is added to the result of matrix multiplication with the values, which is reminiscent of the residual connections used in Transformers (Vaswani et al., 2017). Whereas in 11 the softmax function is used, CMLs use the WTA function, which becomes increasingly similar to softmax for higher values of β. One further salient difference between these models is that CMLs computes the attention vector by combining the affordance g_t and the utilities u_t, see Equ. 9.

Thus altogether, CMLs combine some Transformer-like features with other ingredients that have no correspondence with Transformers. Furthermore, CMLs port these Transformer-like features into a light-weight setting where no offline optimization of parameters through BPTT is required.

Discussion

Planning capability is a key function of the brain, but it is largely open how it is implemented in neural networks of the brain (Mattar and Lengyel, 2022). We have shown that a simple neural network architecture, a Cognitive Map Learner (CML), can learn to plan through self-supervised learning with simple local rules for synaptic plasticity. Surprisingly, its planning performance is in the cases with reversible actions that we tried close to the gold standard for planning in AI: the A* algorithm. However, in contrast to the A* algorithm, the CML does not require that complete knowledge of the environment is provided a-priori, it can learn a cognitive map autonomously through exploration. Furthermore, the CML is a substantially more flexible planner, because it can be applied to any start and goal location without new learning or computational overhead. It is also more suitable for online planning since it can deal intelligently with contingencies, for example, if some action is currently not feasible in some node. Additionally, we have shown that CMLs exhibit generalization capabilities in a spatial 2D navigation task which are beyond the reach of planning with A*: The CML manages to infer spatial relations to find a shortest path to a target location using path segments that it had never encountered during learning. Finally, the CML can be applied also to motor control in high-dimensional action spaces. In fact, it provides a viable alternative to the more common Reinforcement Learning (RL) approach, since it is substantially more flexible. Furthermore, whereas RL in high dimensional state- or action-spaces spaces is plagued by the large number of trials that are needed to learn, the CML just requires a moderate set of exploration trials. This makes it interesting as a new approach towards understanding biological motor control. But it appears to be also of interest as a lightweight but functionally powerful architecture for robots and other edge devices. In fact, the simple synaptic plasticity rules that the CML requires can be easily be implemented in most types of neuromorphic hardware. Furthermore, the multiplications of a learnt matrix with a vector can be implemented very efficiently with one operation through "in-memory computing", where the weights of the matrix are represented by memristors in a cross-bar array.

A key idea of the CML is to create through self-supervised learning a representation of high-dimensional states that are intimately linked to the representation of actions that cause transitions between states. This approach can be seen as an application of the general view expressed in (Buzsáki, 2019), that neural codes for sensory inputs and internal states acquire meaning for an organism by relating them to the outcomes of actions: A key ingredient of the CML is that it induces an action-oriented geometry in high-dimensional state spaces, whereby states move closer together when it turns out that an action causes a transition from one to the other.

One obvious method to improve the performance and application range of the CML is to add fixed nonlinear pre-processing modules, which can be viewed as kernels, liquids, or reservoirs, before the matrices W_q and W_k. This enables the CML to learn nonlinear embeddings of observations into the state space, and a nonlinear transformation from the state space into the planning core, while still requiring only simple synaptic plasticity rules for linear readouts from such pre-processing modules. Another expansion option is to engage two CMLs in parallel for planning in physical environments, where one CML creates abstract space representations as in our demo for a random graph, and the other a state space that takes the 2D or 3D structure of the environment into account. The former representation has the advantage that efficient trajectories to a goal are produced by the CML even if they involve detours in the physical environment. On the other hand, a more spatially oriented representation offers additional generalization capabilities that enable the planning to use never before encountered shortcuts. A hybrid model with two CML modules can therefore combine the best of both approaches. This hybrid architecture gives also rise to an experimentally testable prediction for the brain: It predicts that the brain employs different state representations, say in hippocampal areas and in the PFC, whose role in planning depends on the concrete planning challenge.

We made no effort to optimize the exploration strategy of the CML, we just used actions randomly. But since the exploration strategy is independent of the inner workings of the CML, it can easily be combined with more efficient exploration heuristics, that could for example be driven by curiosity or novelty, that are likely to further reduce the amount of exploration that the CML needs as a basis for efficient planning. Furthermore, learning can also continue online during planning whenever a planned action is executed, thereby automatically improving state representations and planning performance in those regions of the state space where more planning challenges arise.

The CML appears to satisfy many of the constraints that are expected from the organization of planning in the brain (Mattar and Lengyel, 2022). In particular, it shows how close to optimal planning performance can emerge from simple always-on local prediction learning mechanisms. We did not make here an effort to implement the concept of a CML in a network of spiking neurons with synaptic plasticity rules that are commonly considered in computational neuroscience, but this step seems to pose no obstacles. The most obvious experimentally testable hypothesis of the

CML model is that internal state representations of the brain are malleable, more precisely, those state representations become more similar when the individual learns that one state can be reached from the other with one or few actions. One prominent experimental support for this hypothesis was reported for area IT (Li and DiCarlo, 2008) for the case of a saccade as action. Our theory suggests that this effect also takes place in other cortical areas, such as PFC, independently of the type of action. Also commonly found representational drift of neural codes in the brain (Rule, O'Leary and Harvey, 2019; Driscoll, Duncker and Harvey, 2022) can be seen as support for the malleable nature of internal state representations that are inherent for the CML model, but more difficult to explain for RL-based models.

With regard to the question which learning processes of the brain can better be modelled by RL, and which by a CML-like planning process. one has to take into account that model-based learning also proposes that a model of the impact of actions on states is learnt. However, one usually assumes in RL models that state-representations are fixed during learning, whereas the CML model proposes that these are continuously adapted during learning. On the neural implementation level, the CML proposes that some skills can also be learnt without neuromodulators such as dopamine, which are corner-stones of RL-based models, but are not vital for the CML model. Hence from the CML-perspective, dopamine could play a modified role in brain learning, such as shaping motivation or habits. This question can be tested experimentally through inactivation of DA-ergic neurons during learning.

Finally, we have addressed the relation between the CML and Transformers (Vaswani et al., 2017), a learning architecture that is increasingly outpacing traditional neural network models in current AI. The CML contains some ingredients of the Transformer architecture, but is a substantially more light-weight model that does not require complex offline learning processes such as BPTT. Hence it arguably combines some of the simplicity and functional superiority of Transformers with constraints on learning that arise both in the brain and in neuromorphic hardware.

Methods

Details to planning on abstract graphs

In the planning on abstract graphs task every node is encoded with a unique one-hot encoded observation o_t. To explore the different 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.

The affordance gating function f_g was chosen to be a generalised sigmoid function in the form:

$$\hat{\sigma}(x) = \text{clip}\left(\frac{\sigma(x) - \sigma(0)}{\sigma(\alpha) - \sigma(0)}, 0, 1\right), \tag{12}$$

where σ corresponds to the sigmoid function and α is a constant defining the saturation point of the function. During the learning phase, α was chosen to be 1 and during the planning phase α was set to 0.1.

The matrix W_k was initialized with zeros, while the initial values for W_q and W_v were drawn from a Gaussian distribution with $\mu = 0$ and $\sigma = 0.1$, where for W_v $sigma = 1$ was used.

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.

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

Algorithm 1 Dijkstra graph search algorithm

```
Require: Graph $G$
Require: source $S$
Require: target
  for each vertex $v$ in $G$.vertices do
    dist[$v$] $\leftarrow \infty$
    prev[v] $\leftarrow$ null
    add $v$ to $Q$
  end for
  dist[$S$] $\leftarrow 0$
  while $Q$ is not empty do
    $u \leftarrow$ vertex in $Q$ with min dist[$u$]
    remove $u$ from $Q$
    for each neighbor $v$ of $u$ still in $Q$ do
      alt $\leftarrow$ dist[$u$] $+$ $G$.edges($u, v$)
      if alt $<$ dist[$v$] then
        dist[$v$] $\leftarrow$ alt
        prev[$v$] $\leftarrow$ u
      end if
    end for
  end while
  $S \leftarrow$ empty sequence                          ▷ To find the sequence to a target node
  $u \leftarrow$ target
  if prev[$v$] is defined or $u$ = source then
    while $u$ is defined do
      insert $u$ at the beginning of $S$
      $u \leftarrow$ prev[$u$]
    end while
  end if
```

Details to 2D navigation

In this task, a sensory stimulus is implemented as a unique one-hot encoded observation o_t. Two different environment geometries are considered in this task: rectangular and hexagonal environments. An action corresponds to moving from a field to the neighbouring field, hence there are four actions in the rectangular environments while there six actions in the hexagonal environments. In the environment of the 2D navigation task all actions are available to the model in every state. To account for this special case, the affordance gating function f_g was chosen to return the constant value 1 for all inputs.

The initial values for W_k and W_q were set to zeros, while the initial values for W_v were drawn from a Gaussian distribution, with $\mu = 0$ and $\sigma = 1$..

Details to controlling a quadruped

Details to relating actions 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 ± 0.125 were considered. As there are 8 controllable joints, this yields a set with a total of $2^8 = 256$ different combinations of considered torques. 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. The affordance gating function f_g was chosen to be the same as for the planning on abstract graphs task. Furthermore, a single action was applied 10 times in the environment to guarantee a larger change in the observation.

The initial values for all three matrices W_q, W_k and W_v were drawn from a Gaussian distribution, with $\mu = 0$ and $\sigma = 0.1$, where for W_v $\sigma = 1$ was used.

Details to the observation 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 the observation o_t

Details to the computation of the target vector One 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^* can easily be computed by embedding the desired target observation o^* using W_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 this work: 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 model. 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 10 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 ϕ_{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) ϕ_a from ϕ_{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 prompts in the CML on only focusing on changing the current position and not on any other variables from o_t.

Details to the regularization of W_q During the learning process the CML might try to focus on embedding preferably parts of the observation o_t which are easily predictable to minimize the prediction error. An example for components of the observation o_t that is easy to predict are the angles of the joints, while the spatial position of the ant are harder to predict. A problem can therefore arise when the CML assigns the spatial position very little impact on the embedding in state space, as these components would be used to give the CML a target position using o^*. To ensure that important variables for planning are represented in state space, the embedding matrix W_q is regularized such that every column has a minimum norm of 1.

Details to the sampling of actions To allow the CML to explore the environment, actions are sampled randomly, where actions correspond to a torque value applied to the joints of the model of the ant. 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.

Approximating the inverse of W_v

Computing the inverse of W_v is problematic from a biological perspective as well as for neuromorphic hardware applications. Fortunately there are two alternatives to using the inverse matrix.

First, a decent approximation can be found by using the transpose, W_v^T. Conceptually, if the W_v is an orthonormal matrix, the transpose would equal the inverse, which would be the case if all actions have an orthogonal impact on the state space. Another intuition of why the transpose could be useful is that according to Equ. 8, it would result in computing the inner products between the target vector d_t and the vector encoding the impact of an action on the state space. As the inner product can be used to measure similarities, this would result in giving a higher utility to more similar actions, which would be the desired outcome. Using the transpose yielded very similar performances on all graphs planning tasks compared to the inverse. While using the transpose could be a suitable option for neuromorphic hardware it is still problematic from the biological perspective, as it would require symmetric weights in the brain for which there is little evidence.

The second alternative would be to simply introduce a new inverse matrix W_i, which replaces the inverse W_v^{-1}, and optimizing it alongside the other parameters using the following update rule:

$$\Delta W_i(t) = l_i(a_t - u_t)d_t^T, \tag{14}$$

where l_i corresponds to a learning rate. A control experiment on the planning on abstract graphs task using W_i instead of W_v^{-1} yielded a slight decrease in performance, where on average one more node was used in the trajectories.

Details to the optimization process

The loss function used for the optimization of W_q and W_v is the mean squared error of the next state prediction error and can be found in Equ. 15.

$$L_s = \frac{1}{n_s}\sum_{i=1}^{n_s}(s_t[i] - \hat{s}_t[i])^2 \tag{15}$$

For the optimization of W_k the mean squared error of the gating and the action was employed, as can be seen in Equ. 16.

$$L_a = \frac{1}{n_a}\sum_{i=1}^{n_a}(a_t[i] - g_t[i])^2 \tag{16}$$

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. 4, 5, and 6 were accumulated over a single trajectory and applied at the end of the trajectory.

Acknowledgements

We would like to thank Christopher Summerfield, Guozhang Chen, and Yujie Wu 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.

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.

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.

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.

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.

Jonke, Zeno et al. (2017). 'Feedback inhibition shapes emergent computational properties of cortical microcircuit motifs'. In: *Journal of Neuroscience* 37.35, pp. 8511–8523.

Kamada, Tomihisa, Satoru Kawai et al. (1989). 'An algorithm for drawing general undirected graphs'. In: *Information processing letters* 31.1, pp. 7–15.

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.

Khetarpal, Khimya et al. (2020). 'What can I do here? A Theory of Affordances in Reinforcement Learning'. In: *International Conference on Machine Learning.* PMLR, pp. 5243–5253.

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

Li, Nuo and James J DiCarlo (2008). 'Unsupervised natural experience rapidly alters invariant object representation in visual cortex'. In: *science* 321.5895, pp. 1502–1507.

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

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.

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

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

Rule, Michael E, Timothy O'Leary and Christopher D Harvey (2019). 'Causes and consequences of representational drift'. In: *Current opinion in neurobiology* 58, pp. 141–147.

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.

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.

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*.