SEMI-PARAMETRIC TOPOLOGICAL MEMORY FOR NAVIGATION

Anonymous authors
Paper under double-blind review

ABSTRACT

We introduce a new memory architecture for navigation in previously unseen environments, inspired by landmark-based navigation in animals. The proposed semi-parametric topological memory (SPTM) consists of a (non-parametric) graph with nodes corresponding to locations in the environment and a (parametric) deep network capable of retrieving nodes from the graph based on observations. The graph stores no metric information, only connectivity of locations corresponding to the nodes. We use SPTM as a planning module in a navigation system. Given only 5 minutes of footage of a previously unseen maze, an SPTM-based navigation agent can build a topological map of the environment and use it to confidently navigate towards goals. The SPTM-based agent outperforms existing agents with LSTM memory by a large margin.

1 INTRODUCTION

Deep learning (DL) has recently been used as an efficient approach to learning navigation in complex three-dimensional environments. DL-based approaches to navigation can be broadly divided into three classes: purely reactive (Dosovitskiy & Koltun, 2017; Zhu et al., 2017), based on unstructured general-purpose memory such as LSTM (Mnih et al., 2016; Mirowski et al., 2017), and employing a navigation-specific memory structure based on a metric map (Parisotto & Salakhutdinov, 2017; Gupta et al., 2017).

However, extensive evidence from psychology (Gillner & Mallot, 1998; Wang & Spelke, 2002; Foo et al., 2005) suggests that when traversing environments, animals do not rely strongly on metric representations. Instead, they employ a range of specialized navigation strategies of increasing complexity. According to Foo et al. (2005), one such strategy is landmark navigation – "the ability to orient with respect to a known object". Another is route-based navigation that "involves remembering specific sequences of positions". Finally, map-based navigation assumes a "survey knowledge of the environmental layout", but the map need not be metric and in fact it is typically not: "[...] humans do not integrate experience on specific routes into a metric cognitive map for navigation [...] Rather, they primarily depend on a landmark-based navigation strategy, which can be supported by qualitative topological knowledge of the environment."

In this paper, we propose semi-parametric topological memory (SPTM) – a deep-learning-based memory architecture for navigation, inspired by landmark-based navigation in animals. SPTM consists of two components: a non-parametric memory graph G where each node corresponds to a location in the environment, and a parametric deep network R capable of retrieving nodes from the graph based on observations. The graph contains no metric relations between the nodes, only connectivity information. While exploring the environment, the agent builds the graph by appending its observations to the graph and adding shortcut connections based on detected loop closures. The network R is trained to retrieve nodes from the graph based on an observation of the environment. This allows the agent to localize itself in the graph. Finally, we build a complete SPTM-based navigation agent by complementing the memory with a locomotion network L, which allows the agent to move between nodes in the graph. Both R and L networks are trained in self-supervised fashion, without any manual labeling or reward signal.

We evaluate the proposed system and relevant baselines on the task of goal-directed maze navigation in a simulated three-dimensional environment. The agent is instantiated in a previously unseen maze and given a recording of a walk through the maze (images only, no information about actions taken

or ego-motion). Then the agent is initialized at a new location in the maze and has to reach a goal location in the maze, given an image of that goal. To be successful at this task, the agent must represent the maze based on footage it has seen, and effectively utilize this representation for navigation.

The proposed system outperforms baseline approaches by a large margin. Given 5 minutes of maze walkthrough footage, the system is able to build an internal representation of the environment and use it to confidently navigate to various goals within the maze. In all seven test environments the success rate of an SPTM-based agent on goal-directed navigation is higher than that of the best baseline, and on six out of seven labyrinths – at least by a factor of 2. The agent reaches the goals not only more successfully, but also faster than the baselines.

2 RELATED WORK

Navigation in animals has been extensively studied in psychology. In a seminal work, Tolman (1948) introduced the concept of cognitive map – an internal representation of the environment that enables efficient navigation. The existence of cognitive maps and their exact form in animals, including humans, has been debated since. O'Keefe & Nadel (1978) suggested that internal representations take the form of metric maps. More recently, it has been shown that bees (Cartwright & Collett, 1982; Collett, 1996), ants (Judd & Collett, 1998), and rats (Sutherland et al., 1987) rely largely on simple landmark-based mechanisms for navigation. Bennett (1996) and Mackintosh (2002) question the existence of cognitive maps in animals. Gillner & Mallot (1998), Wang & Spelke (2002), and Foo et al. (2005) argue that humans rely largely on landmark-based navigation.

In contrast, navigation systems developed in robotics are typically based on metric maps, constructed using the available sensory information – sonar, LIDAR, RGB-D, or RGB cameras (Elfes, 1987; Thrun et al., 2005; Durrant-Whyte & Bailey, 2006). Particularly relevant to our work are vision-based SLAM (simultaneous localization and mapping) methods, as reviewed by Cadena et al. (2016). These systems provide high-quality maps under favorable conditions, but they need the precise knowledge of the camera parameters, do not deal well with poor imaging conditions, and do not scale well to large environments.

Modern deep learning (DL) methods allow for end-to-end learning of sensorimotor control, directly predicting control signal from high-dimensional sensory observations, such as images (Mnih et al., 2015). DL approaches to navigation vary both in the learning method – reinforcement learning or imitation learning – and in the memory representation. Purely reactive methods (Dosovitskiy & Koltun, 2017; Zhu et al., 2017) lack explicit memory and therefore cannot be expected to perform well on long-distance goal-directed navigation in previously unseen environments. Systems equipped with general-purpose LSTM memory (Mnih et al., 2016; Pathak et al., 2017; Jaderberg et al., 2017; Mirowski et al., 2017) or episodic memory (Blundell et al., 2016; Pritzel et al., 2017) can potentially store information about the environment. However, these systems have not been demonstrated to perform efficient goal-directed navigation in previously unseen environments. Oh et al. (2016) use an addressable memory for first person view navigation in three-dimensional environments. The authors demonstrate that the proposed memory structure allows to generalize to previously unseen environments better than standard LSTM. A crucial difference from our work is that Oh et al. (2016) experiment with relatively small discrete gridworld-like environments, as opposed to large continuous state space in our work.

Most related to our work are DL navigation systems that use specialized map-like representations. Bhatti et al. (2016) augment a DL system with a metric map produced by a standard SLAM algorithm. Parisotto & Salakhutdinov (2017) use a 2D spatial memory that represents a global map of the environment. Gupta et al. (2017) build a 2D multi-scale metric map using the end-to-end trainable planning approach of Tamar et al. (2016). Our method differs from these approaches in that we are not aiming to build a global metric map of the environment, but instead use a topological map. This allows our method to support navigation in a continuous space without externally provided camera poses or ego-motion information.

Although contemporary approaches in robotics are dominated by metric maps, research on topological maps has a long history. Models based on topological maps have been applied both to navigation in simple 2D mazes (Kuipers & Byun, 1991; Meng & Kak, 1993; Schölkopf & Mallot, 1995) and on

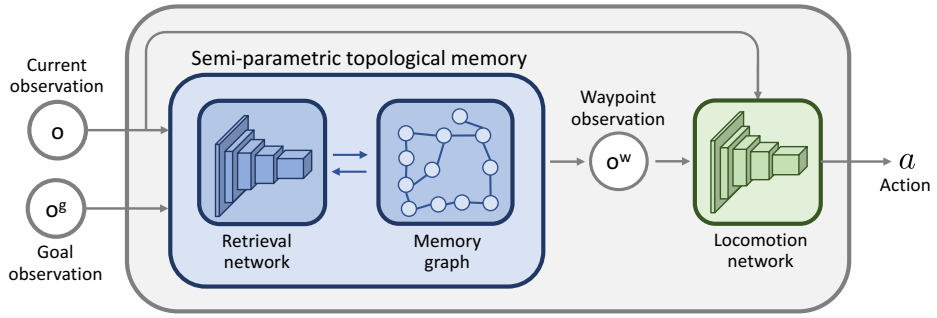


Figure 1: A navigation agent equipped with semi-parametric topological memory (SPTM). Given the inputs – the current observation $\mathbf{o}$ and the goal observation $\mathbf{o}^g$ – SPTM provides a waypoint observation $\mathbf{o}^w$. This waypoint and the current observation $\mathbf{o}$ are fed into the locomotion network L, which outputs the action a to be executed in the environment.

physical systems (Hong et al., 1992; Bachelder & Waxman, 1995; Franz et al., 1998; Thrun, 1998; Fraundorfer et al., 2007). Trullier et al. (1997) provide a review of biologically-inspired navigation systems, including landmark-based ones. Our approach reinterprets this line of work in the context of end-to-end deep learning. In this paper we show that an agent equipped with topological memory can navigate a large realistic three-dimensional maze based on recordings from a monocular camera.

3 METHOD

We consider an agent interacting with an environment in discrete time steps. At each time step t, the agent gets an observation $\mathbf{o}_t$ of the environment and then takes an action a_t from a set of actions $\mathcal{A}$. In our experiments, the environment is a maze in a three-dimensional simulated world, and the observation is provided to the agent as a tuple of several recent images seen by the agent.

The interaction of the agent with a new environment is set up in two stages: exploration and goal-directed navigation. During the first stage, the agent is presented with a recording of a traversal of the environment over a number of time steps T_e, and builds an internal representation of the environment based on this recording. In the second stage, the agent uses this internal representation to reach goal locations in the environment. This goal-directed navigation is performed in an episodic setup, with each episode lasting for a fixed maximum number of time steps or until the goal is reached. In each episode, the goal location is provided to the agent by an observation of this location $\mathbf{o}^g$. The agent has to use the goal observation and the internal representation built during the exploration phase to effectively reach the goal.

3.1 SEMI-PARAMETRIC TOPOLOGICAL MEMORY

We propose a new form of memory suitable for storing internal representations of environments. We refer to it as semi-parametric topological memory (SPTM). It consists of a (non-parametric) memory graph G where each node represents a location in the environment, and a (parametric) deep network R capable of retrieving nodes from the graph based on observations. A high-level overview of an SPTM-based navigation system is shown in Figure 1. Here SPTM acts as a planning module: given the current observation $\mathbf{o}$ and the goal observation $\mathbf{o}^g$, it generates a waypoint observation $\mathbf{o}^w$, which lies on a path to the goal and can be easily reached from the current agent's location. The current observation and the waypoint observation are provided to the locomotion network L responsible for short-range navigation. The locomotion network then guides the agent towards the waypoint, and the loop repeats. Note that both networks R and L are trained in self-supervised fashion, without any externally provided labels or reinforcement signals. We now describe each component of the system in detail.

Retrieval network. The R-network estimates the similarity of two observations $(\mathbf{o}_1, \mathbf{o}_2)$. The network is trained on a set of environments in self-supervised manner, based on trajectories of a

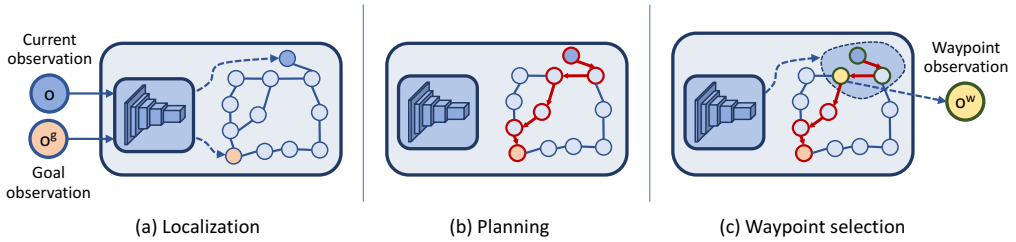


Figure 2: The functioning of semi-parametric topological memory. (a) The retrieval network R localizes in the graph the vertices v^a (blue) and v^g (orange), corresponding to the current agent's observation $\mathbf{o}$ and the goal observation $\mathbf{o}^g$ respectively. (b) The shortest path on the graph between these vertices is computed (red arrows). (c) The waypoint vertex v^w (yellow) is selected as the vertex in the shortest path which is furthest from the agent's vertex v^a, but can still be confidently reached by the agent. The output of SPTM is the corresponding waypoint observation $\mathbf{o}^w = \mathbf{o}_{v^w}$.

randomly acting agent. Conceptually, the network is trained to assign high similarity to pairs of observations that are temporally close, and low similarity to pairs that are temporally distant. In practice, we set this task up as classification: given a pair of observations, the network has to predict if they are temporally close or not.

To generate the training data, we first let a random agent explore the environment, resulting in a sequence of observations $\{\mathbf{o}_1, \ldots \mathbf{o}_N\}$ and actions $\{a_1, \ldots a_N\}$. We then automatically generate training samples from these trajectories. Each training sample is a triple $\langle \mathbf{o}_i, \mathbf{o}_j, y_{ij} \rangle$ consisting of two observations and a binary label. Two observations are considered close ($y_{ij} = 1$) if they are separated by at most $L = 20$ time steps: $|i - j| \leq L$. Negative examples are pairs where the two observations are separated by at least $M \cdot L$ steps, where $M = 5$ is a constant factor defining the margin between positive and negative examples.

We use a siamese architecture for the network R, akin to Zagoruyko & Komodakis (2015). Each of the two input observations is first processed by a deep convolutional encoder based on ResNet-18, which outputs a 512-dimensional embedding vector. These two vectors are concatenated and further processed by a small 5-layer fully-connected network, ending with a 2-way softmax. The network is trained in supervised fashion with the cross-entropy loss. Further details are provided in the supplement.

Memory graph. The graph is populated based on the exploration sequence provided to the agent. Denote the observations in the sequence by $(\mathbf{o}^e_1, \ldots, \mathbf{o}^e_{T_e})$. Each vertex v_i in the graph stores an observation of the environment, $\mathbf{o}_{v_i} = \mathbf{o}^m_i$. Two vertices v_i and v_j are connected by an edge in one of two cases: if they correspond to subsequent time steps, or if the corresponding observations are very close, as judged by the retrieval network R:

$$e_{ij} = 1 \;\Leftrightarrow\; |i - j| = 1 \;\vee\; R(\mathbf{o}_{v_i}, \mathbf{o}_{v_j}) > s_{shortcut}, \tag{1}$$

where $0 < s_{shortcut} < 1$ is a similarity threshold for creating a shortcut connection. The first type of edges corresponds to natural spatial adjacency of the locations, while the second type can be seen as a form of loop closure.

Finding the waypoint. At navigation time, we use SPTM to provide waypoints to the locomotion network. As illustrated in Figure 2, the process includes three steps: localization, planning, and waypoint selection.

In the localization step, the agent localizes itself and the goal in the graph based on its current observation $\mathbf{o}$ and the goal observation $\mathbf{o}^g$, as illustrated in Figure 2 (a). We have experimented with two approaches to localization. In the basic variant, the agent's location is retrieved as the median of $k = 5$ nearest neighbors of the observation in the memory. The siamese architecture of the retrieval network allows for efficient nearest neighbor queries by pre-computing the embeddings of observations in the memory.

An issue with this simple technique is that localization is performed per frame, and therefore the result can be noisy and susceptible to perceptual aliasing – inability to discriminate two locations with similar appearance. We therefore propose a modified approach allowing for temporally consistent self-localization. Namely, we initially perform the nearest neighbor search only in a local neighborhood of the previous agent's localization, and resort to global search in the whole memory only if this initial search fails (that is, the similarity of the retrieved nearest neighbor to the current observation is below a certain threshold s_{local}). This simple modification improves the performance of the method while also reducing the search time.

In the planning step, we find the shortest path on the graph between the two retrieved nodes v^a and v^g, as shown in Figure 2 (b). We used Dijkstra's algorithm in our experiments.

Finally, the third step is to select a waypoint on the computed shortest path, as depicted in Figure 2 (c). We denote the shortest path by

$$\langle v_0^{sp},\, v_1^{sp},\, \ldots\,, v_n^{sp}\rangle\,,\; v_0^{sp} = v^a,\, v_n^{sp} = v^g \tag{2}$$

A naive solution would be to set the waypoint to v_D^{sp}, with a fixed D. However, this heuristic may fail for instance if the graph includes fragments where the agent remains stationary. We therefore follow a more robust adaptive strategy. We choose the furthest vertex along the shortest path that is still confidently reachable:

$$v^w = v_l^{sp}, \quad l = \max_i \{i,\, s.t.\, R(\mathbf{o}, \mathbf{o}_{v_i^{sp}}) > s_{reach}\}, \tag{3}$$

where $0 < s_{reach} < 1$ is a fixed similarity threshold for considering a vertex reachable. The corresponding observation $\mathbf{o}^w = \mathbf{o}_{v^w}$ is the output of the planning process.

3.2 Locomotion Network

The L-network is trained to navigate towards target observations in the vicinity of the agent. The network maps a pair $(\mathbf{o}_1, \mathbf{o}_2)$, consisting of a current observation and a goal observation, into action probabilities: $L(\mathbf{o}_1, \mathbf{o}_2) = \mathbf{p} \in \mathbb{R}^{|\mathcal{A}|}$. The action can then be produced either deterministically as the most probable action, or stochastically as a sample from the distribution. In what follows we use the stochastic policy.

Akin to the retrieval network R, the L-network is trained in self-supervised manner, based on trajectories of a randomly acting agent. Random exploration results in a sequence of observations $\{\mathbf{o}_1, \ldots \mathbf{o}_N\}$ and actions $\{a_1, \ldots a_N\}$. We generate training samples from these trajectories by taking a pair of observations separated by at most $L = 20$ time steps and the action corresponding to the first observation: $((\mathbf{o}_i,\, \mathbf{o}_j),\, a_i)$. The network is trained in supervised fashion on this data, with a softmax output layer and the cross-entropy loss. The architecture of the network is the same as the retrieval network.

Why is it possible to learn a useful controller based on trajectories of a randomly acting agent? The proposed training procedure leads to learning the conditional action distribution $P(a|\mathbf{o}_t, \mathbf{o}_{t+l})$. Even though the trajectories are generated by a random actor, this distribution is generally not uniform. For instance, if $l = 1$, the network would learn actions to be taken to perform one-step transitions between neighboring states. For $l > 1$, training data is more noisy, but there is still useful training signal, which turns out to be sufficient to learn short-range navigation.

3.3 Implementation details

Inputs to the retrieval network R and the locomotion network L are observations of the environment o, represented by stacks of two most recent RGB images obtained from the environment, at resolution 160×120 pixels. Both networks have architectures based on ResNet-18 (He et al.). Note that ResNet-18 is much larger than networks typically used in navigation agents based on reinforcement learning. The use of this high-capacity architecture is made possible by the self-supervised training of our model. To our knowledge, training of a network of this size from scratch with pure reinforcement learning has never been demonstrated.

The training setup is similar for both networks. We generate training data online by executing a random agent in the training environment, and maintain a replay buffer B of most recent samples.

(a) Maze (b) Agent's observation (c) Waypoint observation (d) Goal observation

Figure 3: SPTM-based agent navigating towards a goal in a three-dimensional maze (a). The agent aims to reach the goal, denoted by a star. Given the current agent's observation (b) and the goal observation (d), SPTM produces a waypoint observation (c). The locomotion network is then used to navigate towards the waypoint.

At each training iteration, we sample a mini-batch of training observation pairs at random from the buffer, according to the conditions described in Sections 3.1 and 3.2. We then perform an update using the Adam optimizer (Kingma & Ba, 2015), with learning rate $\lambda = 0.0001$. We train for a total of 1.5 million mini-batch iterations. Further details are provided in the supplement.

We made sure that all operations in SPTM are implemented efficiently. Goal localization is only performed once in the beginning of a navigation episode. Shortest paths to the goal from all vertices of the graph can therefore also be computed once in the beginning of navigation. The only remaining computationally expensive operations are nearest-neighbor queries for agent self-localization in the graph. However, thanks to the siamese architecture of the retrieval network, we can precompute the embedding vectors of observations in the memory and only evaluate the small fully-connected network at test time.

4 EXPERIMENTS

We perform experiments using a simulated three-dimensional environment based on the classic game Doom (Kempka et al., 2016). An illustration of an SPTM agent navigating towards a goal in a maze is shown in Figure 3. We evaluate the proposed method on the task of goal-directed navigation in previously unseen environments and compare it to relevant baselines from the literature.

4.1 SETUP

We are interested in agents that are able to generalize to new environments. Therefore, we used different mazes for training, validation, and testing. We used the same set of textures for all labyrinths, but the maze layouts are very different, and the texture placement is randomized. During training, we used a single labyrinth layout, but created 400 versions with randomized goal placements and textures. In addition, we created 3 mazes for validation and 7 mazes for testing. Layouts of the training and test labyrinths are shown in Figure 4; the validation mazes are shown in the supplement. Each maze is equipped with 4 goal locations, designated by 4 different special objects. The appearance of these special objects is common to all mazes. We used the validation mazes for tuning the parameters of all approaches, and used fixed parameters when evaluating in the test mazes.

The overall experimental setup follows Section 3. When given a new maze, the agent is provided with an exploration sequence of the environment, with the duration of 5 minutes of in-simulation time (equivalent to $10{,}500$ simulation steps). In our experiments, we used sequences generated by a human subject aimlessly exploring the mazes. The same exploration sequences were provided to all algorithms – the proposed method and the baselines. An example sequence from the Val-3 maze is available at `https://youtu.be/4QxO8mdOf3M`.

Given the exploration sequence, the agent attempts a series of goal-directed navigation trials. In each of these, the agent is positioned at a new location in the maze and is presented with an image of the goal location. In our experiments, we used 4 different starting locations, 4 goals per maze, and repeated each trial 6 times (results in these vary due to inevitable stochasticity in the physics of the environment), resulting in 96 trials for each maze. A trial is considered successfully completed if the agent reaches the goal within 5000 simulation steps, or 2.4 minutes of in-simulation time.

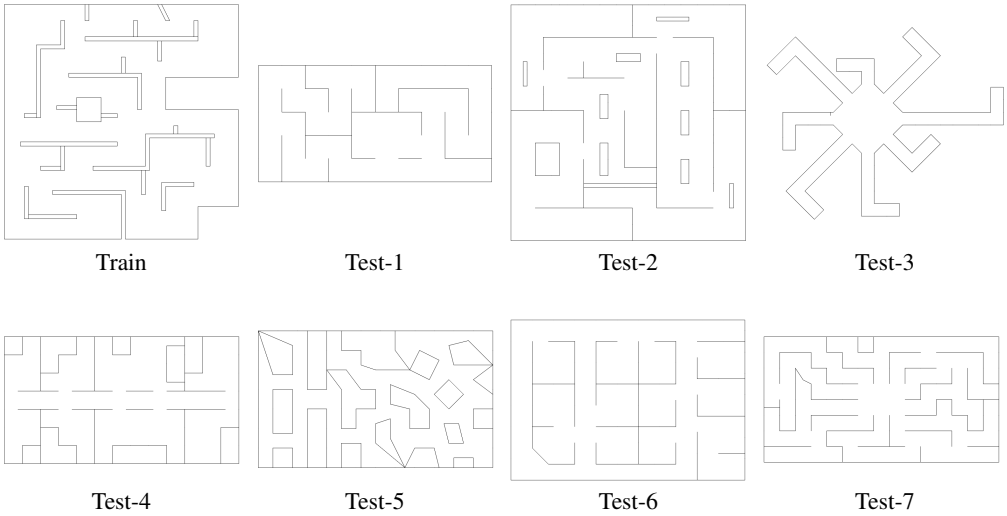


Figure 4: Layouts of training and test mazes.

4.1.1 HYPERPARAMETER TUNING

We select the hyperparameters of the SPTM-based agent based on an evaluation on the validation set, as reported in Table S1 in the supplement. We find that the method performs well for a range of hyperparameter values. Interestingly, the approach is robust to temporal subsampling of the walkthrough sequence. Therefore, in subsequent experiments we subsample the walkthrough sequence by a factor of 4 when building the SPTM graph, unless mentioned otherwise. Another important parameter is the threshold $s_{shortcut}$ for creating shortcuts in the graph. We select this threshold as a percentile of the set of all pairwise distances between observations in the memory, or, in other words, as the desired number of shortcuts to be created. We set this number to 1000 in what follows. We set the threshold values to $s_{local} = 0.7$ and $s_{reach} = 0.9$.

4.2 BASELINES

We compare the proposed method to a set of baselines, representative of the state of the art in deep-learning-based navigation. Note that we study an agent operating in a continuous state space, with no access to ground-truth information such as depth maps or ego-motion. This realistic setup excludes several existing works from our comparison: the full model of Mirowski et al. (2017) that uses ground-truth depth maps and ego-motion, the method of Gupta et al. (2017) that operates on a discrete grid given ground-truth ego-motion, and the approach of Parisotto & Salakhutdinov (2017) that requires the knowledge of ground-truth global coordinates of the agent.

The first baseline is a goal-agnostic agent without memory. The agent is not informed about the goal, but may reach it by chance. We train this network in the training maze using asynchronous advantage actor-critic (A3C) (Mnih et al., 2016). The agent is trained on the surrogate task of collecting invisible beacons around the labyrinth. (The beacons are made invisible to avoid providing additional visual guidance to the agents.) At the beginning of each episode, the labyrinth is populated with 1000 of these invisible beacons, at random locations. The agent receives a reward of 1 for collecting a beacon and 0 otherwise. Each episode lasts for $3{,}150$ simulation steps. We train the agent with the A3C algorithm and use an architecture similar to Mnih et al. (2016). Further details are provided in the supplement.

The second baseline is a feedforward network trained on goal-directed navigation, similar to Zhu et al. (2017). The network gets its current observation, as well as an image of the goal, as input. It gets the same reward as the exploration agent for collecting invisible beacons, but in addition it gets a large reward of 500 for reaching the goal. This network can go towards the goal if the goal is within its field of view, but it lacks memory, so it is fundamentally unable to make use of the exploration phase. The network architecture is the same as in the first baseline, but the input is the concatenation of the 4 most recent frames and the goal image.

	Test-1	Test-2	Test-3	Test-4	Test-5	Test-6	Test-7	Avg. rank
Goal-agnostic feedforward	26	25	27	23	32	20	20	3.7
Goal-agnostic LSTM	53	39	45	18	27	36	19	2.7
Goal-seeking feedforward	30	24	29	18	24	27	22	3.7
Goal-seeking LSTM	34	27	15	25	16	33	4	3.7
Ours	**100**	**84**	**90**	**89**	**76**	**85**	**60**	**1**

Table 1: Comparison of the SPTM-based agent with baseline approaches. We report the percentage of navigation trials successfully completed in 5000 steps (higher is better). We also report the average rank of each method (lower is better).

The third and fourth baseline approaches are again goal-agnostic and goal-directed agents, but equipped with LSTM memory. The goal-directed LSTM agent is similar to Mirowski et al. (2017). When training these networks, we do not switch the goal locations in every episode, but instead repeat each episode 5 times before advancing to the next one. We do not reset the state of the memory cells between these episodes. This way, the agent can learn to store the location of the goals in its memory and use it for efficient navigation. At test time, we feed the exploration sequence to the LSTM agent and then let it perform goal-directed navigation without resetting the LSTM state.

4.3 RESULTS

Table 1 shows, for each test maze, the percentage of navigation trials successfully completed within 5000 steps, equivalent to 2.4 minutes of real-time simulation. Figure 5 presents the results on the test mazes in more detail, by plotting the percentage of completed episodes as a function of the trial duration.

The proposed SPTM-based agent is superior to the baseline approaches on all mazes. As Table 1 demonstrates, in all mazes except the simplest one (Test-1), it is at least twice as successful as the best performing baseline. Figure 5 demonstrates that the proposed approach is not only successful overall, but that the agent typically reaches the goals faster than the baseline methods.

The difference in the performance between feedforward and LSTM baseline variants is generally small and not consistent across mazes. This suggests that standard LSTM memory is not sufficient to efficiently make use of the provided walkthrough footage. One reason can be that recurrent networks, including LSTMs, struggle with storing long sequences (Goodfellow et al., 2016). The duration of the walkthrough footage, $10{,}000$ time steps, is beyond the capabilities of standard recurrent networks. SPTM is at an advantage, since it stores all the provided information by design.

Why is the performance of the baseline approaches in our experiments significantly weaker than reported previously (Mirowski et al., 2017)? The key reason is that we study generalization of agents to previously unseen environments, while Mirowski et al. (2017) train and evaluate agents in the same environment. The generalization scenario is much more challenging, but also more realistic. A mobile agent typically does not have an opportunity to spend days or weeks exploring a new environment and needs to quickly adapt to new conditions. Our results show that existing methods struggle with this type of generalization.

Figure 6 (left) shows a trajectory of a walkthrough provided to the algorithms in the Val-3 maze. The shortcut connections made automatically in the SPTM graph are marked in red. We selected a conservative threshold for making shortcut connections to ensure there are no false positives. Still, the automatically discovered shortcut connections greatly increase the connectivity of the graph: after introducing the shortcut connections, the average length of the shortest path to the goal, computed over all nodes in the graph, drops from $2{,}500$ to 772 steps.

Figure 6 (right) demonstrates three example trajectories of the SPTM agent performing goal-directed navigation. In successful trials, the agent deliberately goes for the goal, making use of the environment representation stored in SPTM. The rightmost trajectory shows a failure case, where the agent cannot successfully reach the goal. This happens because purely vision-based self-localization becomes increasingly difficult in large environments.

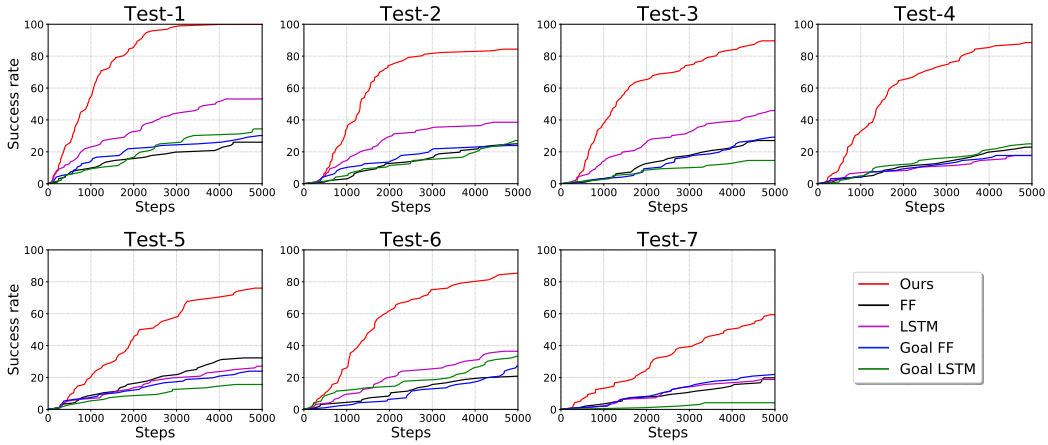
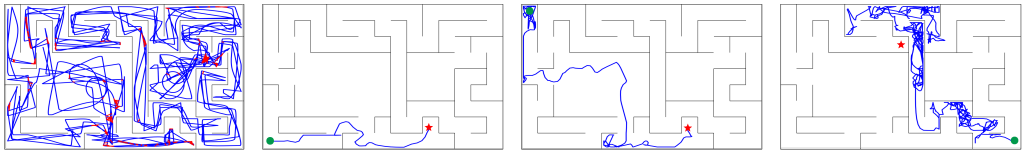


Figure 5: Percentage of successful navigation trials as a function of trial duration. Higher is better.



Walkthrough | Track 1 | Track 2 | Track 3

Figure 6: A walkthrough trajectory (left) and three goal-directed navigation tracks (right) in the Val-3 maze. In the walkthrough trajectory, the shortcuts automatically found in the SPTM graph are shown in red. Goal-directed navigation trials shown in Tracks 1 and 2 have been successful, in Track 3 - not. Start positions are shown in green, goals – in red.

Table 2 shows an ablation study of the SPTM-based navigation agent on the validation set. Removing vision-based shortcuts from the graph leads to a dramatic decline in performance. The agent with independent per-frame localization performs well on two out of three mazes, but underperforms on the more challenging Val-3 maze. A likely explanation is that perceptual aliasing gets increasingly problematic in larger mazes.

	Val-1	Val-2	Val-3
Ours - no visual shortcuts	68	47	28
Ours - per frame localization	**100**	**84**	50
Ours - full	**100**	**84**	**97**

Table 2: Ablation study of the SPTM-based navigation agent. We report the percentage of navigation trials successfully completed in 5000 steps in validation mazes (higher is better).

5 Conclusion

We have proposed semi-parametric topological memory (SPTM) – a memory architecture consisting of a non-parametric component – a topological graph, and a parametric component – a deep network capable of retrieving nodes from the graph given observations of the environment. We have shown that SPTM can act as a planning module in a navigation system. This navigation agent can efficiently reach goals in a previously unseen environment after being presented with only 5 minutes of maze traversal footage. We see several avenues of future work. First, improving the performance of R and L networks will directly improve the overall quality of the system. Second, while the current system explicitly avoids using ego-motion information, it is known from psychology that noisy ego-motion estimation and path integration are useful navigation mechanisms. Integrating these in our model would further improve its robustness. Third, in our current system the size of the memory grows linearly with the duration of the exploration period. This may become problematic when navigating in very large environments, or in scenarios such as lifelong learning. A possible solution is adaptive subsampling of the memory, by only keeping the most informative or discriminative observations. Finally, it would be interesting to integrate SPTM into an end-to-end trainable system.

REFERENCES

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, et al. TensorFlow: A system for large-scale machine learning. In *OSDI*, 2016.

Ivan A. Bachelder and Allen M. Waxman. A view-based neurocomputational system for relational map-making and navigation in visual environments. *Robotics and Autonomous Systems*, 16(2):267 – 289, 1995.

A T Bennett. Do animals have cognitive maps? *Journal of Experimental Biology*, 199(1):219–224, 1996.

Shehroze Bhatti, Alban Desmaison, Ondrej Miksik, Nantas Nardelli, N. Siddharth, and Philip H. S. Torr. Playing doom with slam-augmented deep reinforcement learning. *arxiv*, abs/1612.00380, 2016.

Charles Blundell, Benigno Uria, Alexander Pritzel, Yazhe Li, Avraham Ruderman, Joel Z. Leibo, Jack Rae, Daan Wierstra, and Demis Hassabis. Model-free episodic control. *arxiv*, abs/1606.04460, 2016.

Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, José Neira, Ian D. Reid, and John J. Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Trans. Robotics*, 32(6):1309–1332, 2016.

B. A. Cartwright and T. S. Collett. How honey bees use landmarks to guide their return to a food source. *Nature*, 295, 1982.

François Chollet et al. Keras. `https://github.com/fchollet/keras`, 2015.

T S Collett. Insect navigation en route to the goal: multiple strategies for the use of landmarks. *Journal of Experimental Biology*, 199(1):227–235, 1996.

Alexey Dosovitskiy and Vladlen Koltun. Learning to act by predicting the future. In *ICLR*, 2017.

Hugh F. Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: part I. *IEEE Robot. Automat. Mag.*, 13(2):99–110, 2006.

Alberto Elfes. Sonar-based real-world mapping and navigation. *IEEE J. Robotics and Automation*, 3(3):249–265, 1987.

Patrick Foo, William H. Warren, Andrew Duchon, and Michael J. Tarr. Do humans integrate routes into a cognitive map? Map- versus landmark-based navigation of novel shortcuts. *Journal of Experimental Psychology*, 31(2):195–215, 2005.

Matthias O. Franz, Bernhard Schölkopf, Hanspeter A. Mallot, and Heinrich H. Bülthoff. Learning view graphs for robot navigation. *Autonomous Robots*, 5(1):111–125, 1998.

Friedrich Fraundorfer, Christopher Engels, and David Nistér. Topological mapping, localization and navigation using image collections. In *IROS*, pp. 3872–3877. IEEE, 2007.

Sabine Gillner and Hanspeter A. Mallot. Navigation and acquisition of spatial knowledge in a virtual maze. *Journal of Cognitive Neuroscience*, 10(4):445–463, 1998.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

Saurabh Gupta, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive mapping and planning for visual navigation. In *CVPR*, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*.

J. Hong, X. Tan, B. Pinette, R. Weiss, and E. M. Riseman. Image-based homing. *IEEE Control Systems*, 12(1): 38–45, 1992.

Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z. Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. In *ICLR*, 2017.

S. P. D. Judd and T. S. Collett. Multiple stored views and landmark guidance in ants. *Nature*, 1998.

Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. ViZDoom: A Doom-based AI research platform for visual reinforcement learning. In *IEEE Conference on Computational Intelligence and Games*, 2016.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

Benjamin Kuipers and Yung-Tai Byun. A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations. *Robotics and Autonomous Systems*, 8(1):47 – 63, 1991.

N. J. Mackintosh. Do not ask whether they have a cognitive map, but how they find their way about. *Psicologica*, pp. 165–185, 2002.

Min Meng and Avinash C. Kak. NEURO-NAV: A neural network based architecture for vision-guided mobile robot navigation using non-metrical models of the environment. In *ICRA*, 1993.

Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andrew J. Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, Dharshan Kumaran, and Raia Hadsell. Learning to navigate in complex environments. In *ICLR*, 2017.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540), 2015.

Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *ICML*, 2016.

Junhyuk Oh, Valliappa Chockalingam, Satinder P. Singh, and Honglak Lee. Control of memory, active perception, and action in minecraft. In *ICML*, 2016.

John O'Keefe and Lynn Nadel. *The Hippocampus as a Cognitive Map*. Oxford: Clarendon Press, 1978.

Emilio Parisotto and Ruslan Salakhutdinov. Neural map: Structured memory for deep reinforcement learning. *arxiv*, abs/1702.08360, 2017.

Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *ICML*, pp. 2778–2787, 2017.

Alexander Pritzel, Benigno Uria, Sriram Srinivasan, Adrià Puigdomènech Badia, Oriol Vinyals, Demis Hassabis, Daan Wierstra, and Charles Blundell. Neural episodic control. In *ICML*, volume 70, 2017.

B. Schölkopf and HA. Mallot. View-based cognitive mapping and path planning. *Adaptive Behavior*, 3(3): 311–348, 1995.

R. J. Sutherland, G. L. Chew, J. C. Baker, and R. C. Linggard. Some limitations on the use of distal cues in place navigation by rats. *Psychobiology*, 15(1):48–57, Mar 1987.

Aviv Tamar, Sergey Levine, Pieter Abbeel, Yi Wu, and Garrett Thomas. Value iteration networks. In *NIPS*, 2016.

Sebastian Thrun. Learning metric-topological maps for indoor mobile robot navigation. *Artif. Intell.*, 99(1): 21–71, 1998.

Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. The MIT Press, 2005.

Edward C. Tolman. Cognitive maps in rats and men. *Psychological Review*, 55(4):189–208, 1948.

Olivier Trullier, Sidney I. Wiener, Alain Berthoz, and Jean-Arcady Meyer. Biologically based artificial navigation systems: review and prospects. *Progress in Neurobiology*, 51(5):483 – 544, 1997.

Ranxiao Frances Wang and Elizabeth S Spelke. Human spatial representation: insights from animals. *Trends in Cognitive Sciences*, 6(9):376 – 382, 2002.

Sergey Zagoruyko and Nikos Komodakis. Learning to compare image patches via convolutional neural networks. In *CVPR*, 2015.

Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J. Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *ICRA*, 2017.

SUPPLEMENTARY MATERIAL

S1 METHOD DETAILS

We used the similarity threshold $s_{reach} = 0.9$ when checking if a vertex in the SPTM graph is reachable form the current agent's observation. For creating shortcut connections in the graph, we used a conservative threshold equal to the 99.99-th percentile of the set of all pairwise similarities between observations stored in the graph.

S1.1 NETWORK ARCHITECTURES

The retrieval network R and the locomotion network L are both based on ResNet-18 He et al.. Both take 160×120 pixel images as inputs. The networks are initialized as proposed by He et al.. We used an open ResNet implementation `https://github.com/raghakot/keras-resnet/blob/master/resnet.py`.

The R-network accepts two observations as input, and first feeds each of them through a convolutional encoder with ResNet-18 architecture. Each of the encoders produces a 512-dimensional embedding vector. These are concatenated and fed through a fully-connected network with 4 hidden layers with 512 units each and ReLU non-linearities.

The L-network also accepts two observations, but in contrast with the R network processes them jointly, after concatenating them together. A ResNet-18 convolutional encoder is followed by a single fully-connected layer with 7 outputs and a softmax. The 7 outputs correspond to all available actions: do nothing, move forward, move backward, move left, move right, turn left, turn right.

S1.2 TRAINING

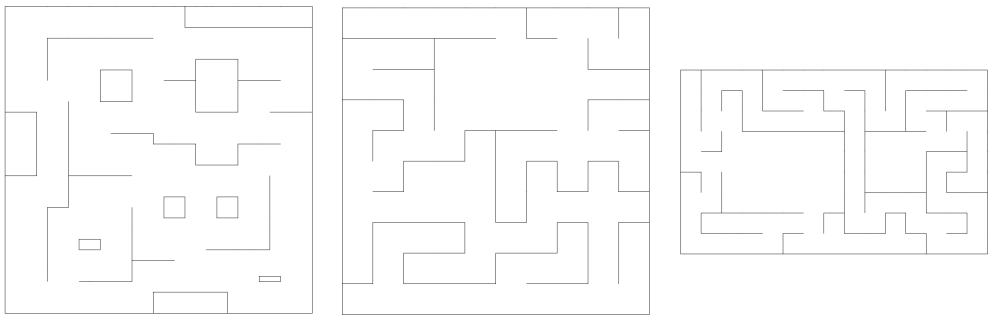
We implemented the training in Keras (Chollet et al., 2015) and Tensorflow (Abadi et al., 2016). The training setup is similar for both networks. We generate training data online by executing a random agent in the environment, and maintain a replay buffer B of the size $|B| = 10{,}000$. We run the random agent for $10{,}000$ steps and then perform 50 mini-batch iterations of training. For the random agent, as well as for all other agents, we use action repeat of 4 – that is, every selected action is repeated 4 times. At each training iteration, we sample a mini-batch of 64 training observation pairs at random from the buffer, according to the conditions described in Sections 3.2 and 3.1. We then perform an update using the Adam optimizer (Kingma & Ba, 2015), with learning rate $\lambda = 0.0001$, momentum parameters $\beta_1 = 0.9$ and $\beta_2 = 0.999$, and the stabilizing parameter $\varepsilon = 10^{-8}$. We train for a total of 1.5 million mini-batch iterations.

S2 BASELINE DETAILS

The baselines are based on an open A3C implementation `https://github.com/pathak22/noreward-rl`. We have used the architecture of Mnih et al. (2016) and Mirowski et al. (2017). The feedforward model consists of two convolutional layers and two fully-connected layer, from which the value and the policy are predicted. In the LSTM model the second fully connected layer is replaced by LSTM. The input to the networks is a stack of 4 most recent observed frames, in grayscale, resized to 84×84 pixels. We have trained the baselines for 50 million action steps, which corresponds to 200 million simulation steps because of action repeat. We selected the snapshot to be used at test time based on the training reward.

S3 ADDITIONAL RESULTS

Layouts of the validation mazes are shown in Figure S1. Plots of success rate as a function of trial duration on the validation mazes are shown in Figure S2. Performance of an SPTM-based agent with varying hyperparameters is shown in Table S1.



Val-1 Val-2 Val-3

Figure S1: Layouts of the mazes used for validation.

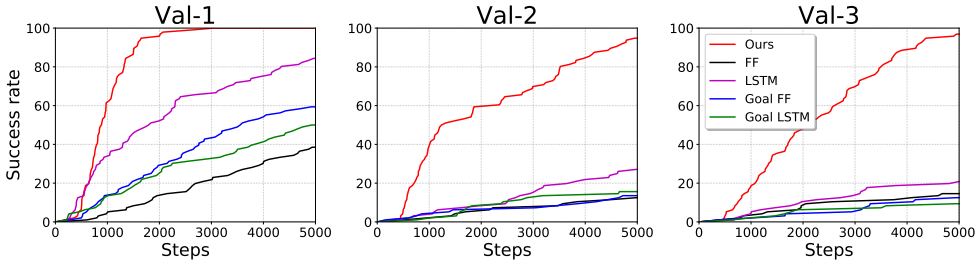


Figure S2: Percentage of successful navigation trials as a function of trial duration, in the validation mazes. Higher is better.

Parameters					Environment		
Localization	#shortcuts	Mem. subsamp.	s_{local}	s_{reach}	Val-1	Val-2	Val-3
Per frame	4000	1	-	0.9	100	84	39
Smooth	4000	1	0.7	0.9	99	73	64
Per frame	4000	2	-	0.9	100	90	63
Smooth	4000	2	0.7	0.9	100	88	97
Smooth	4000	1	0.7	0.9	99	73	64
Smooth	4000	2	0.7	0.9	100	88	97
Smooth	4000	4	0.7	0.9	71	56	10
Smooth	0	1	0.7	0.9	74	38	45
Smooth	2000	1	0.7	0.9	98	63	50
Smooth	4000	1	0.7	0.9	99	73	64
Smooth	8000	1	0.7	0.9	99	94	90
Smooth	4000	1	0.7	0.9	99	73	64
Smooth	2000	2	0.7	0.9	98	89	92
Smooth	1000	4	0.7	0.9	100	95	97
Smooth	500	8	0.7	0.9	44	46	16
Smooth	4000	1	0.7	0.9	99	73	64
Smooth	4000	1	0.6	0.9	99	77	63
Smooth	4000	1	0.8	0.9	98	77	57
Smooth	4000	1	0.7	0.8	98	71	51
Smooth	4000	1	0.7	0.95	100	78	68

Table S1: Analysis of hyperparameter importance on the validation set. We report the percentage of navigation trials successfully completed in 5000 steps (higher is better).