Memory-Augmented Reinforcement Learning for Image-Goal Navigation

Lina Mezghani, Sainbayar Sukhbaatar, Thibaut Lavril, Oleksandr Maksymets, Dhruv Batra, Piotr Bojanowski, Karteek Alahari

Memory-Augmented Reinforcement Learning for Image-Goal Navigation

Lina Mezghani[1,2] Sainbayar Sukhbaatar[1] Thibaut Lavril[1] Oleksandr Maksymets[1]
Dhruv Batra[1,3] Piotr Bojanowski[1] Karteek Alahari[2]

[1]Facebook AI Research [2]Inria* [3]Georgia Institute of Technology

Abstract

In this work, we address the problem of image-goal navigation in the context of visually-realistic 3D environments. This task involves navigating to a location indicated by a target image in a previously unseen environment. Earlier attempts, including RL-based and SLAM-based approaches, have either shown poor generalization performance, or are heavily-reliant on pose/depth sensors. We present a novel method that leverages a cross-episode memory to learn to navigate. We first train a state-embedding network in a self-supervised fashion, and then use it to embed previously-visited states into an agent's memory. In order to avoid overfitting, we propose to use data augmentation on the RGB input during training. We validate our approach through extensive evaluations, showing that our data-augmented memory-based model establishes a new state of the art on the image-goal navigation task in the challenging Gibson dataset. We obtain this competitive performance from RGB input only, without access to additional sensors such as position or depth.

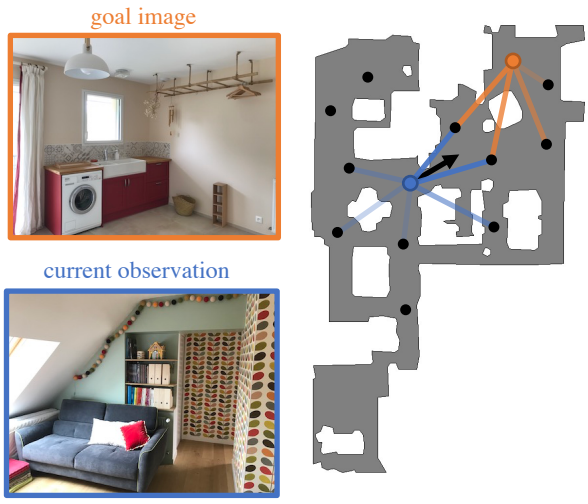


Figure 1. We tackle the problem of image-goal navigation. The agent (shown as the blue dot) is given an image from a goal location (orange dot) which it must navigate to. To address this task, our agent stores a cross-episode memory of previously visited states (black dots), and uses a navigation policy that puts attention (lines) on this memory.

1. Introduction

Embodied vision is an emerging field that tackles computer vision problems coupled to control. Issues considered in this context go beyond the classical computer-vision setup of learning from fixed-size pre-defined datasets. This results in several interesting tasks, such as low-level control point-goal navigation [32], object-goal navigation [2] or even tasks requiring natural language understanding, e.g., embodied question answering [13]. In this work, we focus on the problem of image-goal navigation [37], wherein an agent has to learn to navigate to a location, which is specified by observations taken from there. In terms of complexity this task lies in between those of point-goal and object-goal navigation. Indeed, it does not require learning the association between visual inputs and manual labels, but it also needs a higher-level understanding of scenes for navigating through them, guided by relative positions. This task can be deemed the embodied equivalent of the classical instance-level recognition problem in images.

There are several facets to this task, which make it challenging. The primary requirement in accomplishing this task is a high-level understanding of the surrounding scene. The agent needs to build an internal representation of the surroundings such that the location of the goal can be identified. Once the relative location of the goal is determined, the agent must learn a policy to reach it. For goals that are nearby, this implies issuing actions to finely adapt the viewpoint. On the other hand, for far-away goals, the agent needs to design a high-level navigation plan. And finally, in cases where the goal is not within sight, this task also requires an emergent exploration behaviour.

*Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LJK, 38000 Grenoble, France

In order to have a fully end-to-end learning setup, in this work we propose to tackle the image-goal navigation problem with reinforcement learning. Our task now amounts to training a goal-conditioned policy. Training such a policy with very high-dimensional visual inputs obtained from photorealistic simulation further adds to the challenge. Previous work has either shown results in a limited setting with synthetic data [15] or reported poor RL-based performance [9]. Following Chaplot *et al*. citechaplot2020neural, and in contrast to some previous works [22, 37], we aim to move away from such limited setups and target unseen environments. This requires generalization from the agent's policy, for which there are no guarantees or known cooking recipes. The task is made especially difficult due to the limited size of the available datasets [34], which leads existing RL-based solutions to overfit.

We use a distributed implementation of proximal policy optimization [32] to learn a policy for image-goal navigation in an end-to-end fashion. As opposed to previous work, we train our goal-conditioned policy based on panoramic RGB inputs only, ignoring additional sensors such as the agent's position or depth maps. Due to the aforementioned high complexity of visual inputs, and the additional challenge of generalizing to novel scenes, we explore efficient policy regularization. To this end, we use effective computer vision data-augmentation techniques, such as random cropping and color-space distortions. In order to properly exploit information from past experiences, we endow the agent with a memory, and condition the policy on it using an attention-based network, akin to the Transformer architecture [31]. We empirically evaluate our approach on the challenging image-goal navigation task on the Gibson [34] dataset, following Chaplot *et al*. [9]. To further validate the effectiveness of our approach, we extend the previous evaluation dataset by generating a harder variant of the task and show strong performance on it.

Contributions.: **(i)** We show that the challenging problem of image-goal navigation can be successfully tackled with reinforcement learning, despite the high-dimensional inputs, **(ii)** we show that proper regularization and hence better generalization can be obtained by using effective computer vision data augmentation strategies, and finally, **(iii)** we propose and empirically validate a novel memory-based attention mechanism leading to state-of-the-art performance on the task.

2. Related Work

SLAM-based Methods. The task of navigation has been studied in the context of simultaneous localization and mapping (SLAM) in robotics [29]. Several SLAM methods comprise multiple hand-crafted modules to address strictly-defined problems in specific environments [12, 25, 30]. These modules have been progressively replaced with learning-based functions: some approaches [8] implement the localization module with a neural network, while others [16] replace the metric map with a latent mapping of the environment. Variants of latent mapping also include a topological map whose nodes contain geometric and semantic information about the environment, as well as a global planner that relies on it [9]. Other works replace SLAM entirely by deep models without explicit planning, and instead rely on a map or memory structure [1, 26, 36]. The major drawback of such methods is that they contain multiple modules that are often trained in a supervised fashion, requiring a large amount of annotated data. Even when the labelling process can be automatized, e.g., with a simulator, these approaches remain heavily-reliant on the nature of the simulator they were crafted for.

RL-based Navigation. Another popular class of methods involves training deep models with reinforcement learning to solve navigation tasks without an explicit world representation. They use end-to-end frameworks with modules that are less hand-crafted than SLAM-based methods, and have shown good performance on synthetic mazes [24] as well as real-world data [5, 23]. Such methods have also been explored on indoor-scenes datasets, similar to our setup on image-goal [37] and object-goal [35] navigation tasks. These works use an actor-critic model whose policy is a function of both the target and the current state, and show generalization results across targets. However, generalization across scenes and environments is still a major issue of RL-based methods, as well as sample-inefficiency, which represents a significant drawback for robotics applications.

Combining RL and Planning. A few recent works have augmented RL-based methods with topological structures, like graphs [3, 10, 22, 27, 33] or memory buffers [4, 15, 20]. They store representations of the visited locations and exploit them at navigation time. The process of building these representations can be done offline [3, 22, 27], and requires human-generated data in some cases [27]. For example, the test phase in [27] contains a warm-up stage where the agent builds a graph memory from human trajectories. Alternatives to this manual annotation do exist, such as building a graph directly with reinforcement learning, using the value function of a goal-conditioned policy as edges weights [14], or a buffer of past observations and attending to them [15]. For now, these methods have only been tested on synthetic datasets, and have not proven to be scalable to high-dimensional visually-realistic setups. In the context of other RL tasks, data augmentation has recently been explored as a regularizer for pixel-based RL for DM Control and Atari [19, 21]. Although data augmentation has been used in robotics applications [18], we show that it helps in planning setups such as navigation in Gibson [34].

Exploration and Representation Learning. Closely re-

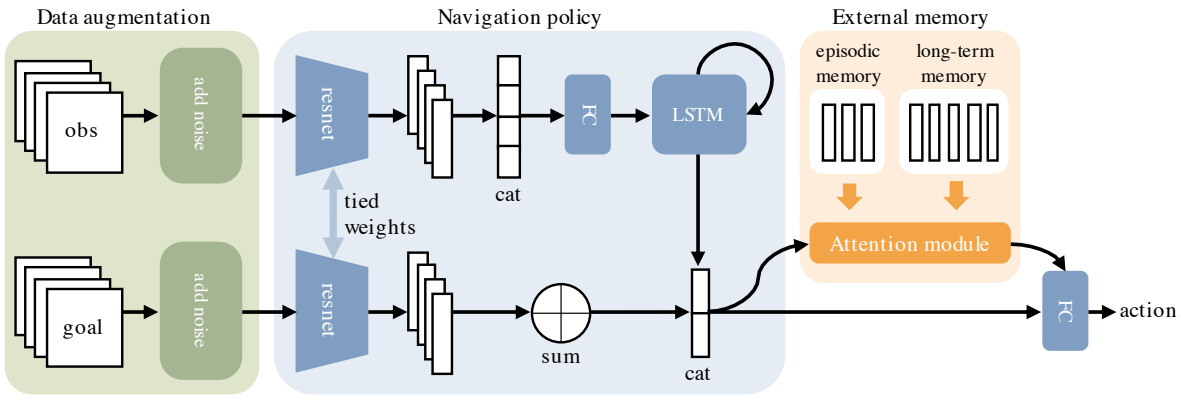


Figure 2. An overview of our agent model that consists of three parts: data augmentation for better generalization, a navigation policy for picking actions, and an external memory for conditioning on previous observations.

lated to navigation, the task of exploration has also been extensively studied and has led to interesting breakthroughs in representation learning. In particular, learning to explore unseen environments has been seen through the spectrum of computer vision [7, 17], SLAM-based [6], and RL-based [11] approaches. Methods such as [22, 28] leverage a self-supervised representation learning stage to prepare the exploration phase. This pretraining stage allows to learn task-agnostic representations of the locations in the environment so as to reduce the dimensionality of the problem. For these works, the generalizability of the learned representations to unseen environments has not been established. Moreover, the use of these representations for navigation tasks was only limited to single-environment setups. Our work extends this line of study by first showing that a self-supervised pretraining phase allows to learn useful information that generalizes to unseen environments, as well as proposing a novel attention-based navigation policy that takes advantage of this information.

3. Problem Formulation

At the beginning of a navigation episode, an agent is given a target observation x^*, composed of an RGB image from the target location. At each timestep t, the agent performs an action a_t and receives the next observation x_{t+1} as well as a reward r_t from the environment. The objective is to learn a navigation policy function $\pi(a_t|x_t, x^*)$ that brings the agent closer to the target location. We complete the definition of our setup with the following details.

Action Space. It comprises 4 actions: `MOVE_FORWARD`, `TURN_LEFT`, `TURN_RIGHT` and `STOP`. Please refer to Section 5.1 for numerical details.

Success Criterion. An episode is considered successful if the agent performs the stop action within a range of l from the target location. In cases where the agent performs the stop action outside of this range, or if the maximum number of steps is exceeded before the agent performs the stop action, the episode is considered a failure.

Observation Space. The observation of the agent x_t as well as the goal observation x^* are the RGB images of the first-person view at those locations. Each RGB image is a panoramic sensor of size $v \times 3 \times 128 \times 128$. We compute this panoramic input by gathering observations from v successive rotations of angle $(360/v)°$ from our agent's location. Note that we do not have access to neither the agent's position nor any depth sensor information.

Reward. We follow the classic setup for image-goal navigation where the reward is split into three components: (i) *sparse success reward*: that rewards the agent for performing the stop action within the success range around the target location, (ii) *dense shaping reward*: that is equal to the decrease in distance to the goal, (iii) *dense slack reward*: that penalizes the agent for being alive at each step, and encourages shorter trajectories.

4. Approach

We now describe our approach for solving the image-goal navigation task. As shown in Figure 2, our model has three components: a data augmentation module to add visual complexity and hence improve generalization, a navigation policy that learns to pick appropriate actions, and an external memory for leveraging past experiences.

4.1. Data Augmentation

To improve the generalization capacity of our agent to unseen environments, we apply transformations on the observations of the simulator by using classic data augmentation techniques. We use two kinds of data transformations: (i) **random cropping** that increases the input image size and takes a random crop of the original size in it, and (ii) **color jitter** that randomly changes the brightness, contrast, saturation and hue levels of the image. An illustration for

Figure 3. Illustration of data augmentation that we use to train our model. We consider both color jittering (left) and random crops (right). For a panoramic observation with v views, the parameters of the augmentation are sampled independently.

these transformations is shown in Figure 3. At navigation time, the agent receives the current and the goal observations from the simulator at each timestep. We apply both transformations sequentially to each of the v views of the current and goal observations independently. This process allows for more visual diversity in the training data.

4.2. Navigation Policy

Once the current and goal observations pass through the data augmentation phase, we use them in the navigation policy module, which computes a probability distribution over all possible actions: $\pi(a_t|x_t, x^*)$.

First, the policy encodes each observation separately, as shown in Figure 2. We encode the current observation by feeding each of the v views separately to the same convolutional neural network. The v vectors resulting from this operation are concatenated and passed into a fully-connected network to reduce their dimension. This reduced-dimension output is then fed into an LSTM along with a representation of the previous action, and its resulting vector w_t^{obs} represents the embedding of the observation at timestep t.

To encode the goal observation x^*, we process it through the same convolutional neural network as the current observation. However, the outputs corresponding to the different views are added together instead of being concatenated, in order to make the representation of the goal location rotation-invariant. We denote by w_t^{goal} the resulting feature vector at timestep t.

Next, we make a joint representation by concatenating the current and goal feature vectors, before passing it through a fully-connected network to output an action:

$$w_t^{\text{joint}} = \texttt{cat}(w_t^{\text{obs}}, w_t^{\text{goal}}), \tag{1}$$

$$\pi(a_t|x_t, x^*) = \texttt{FC}(w_t^{\text{joint}}). \tag{2}$$

4.3. External Memory

We then add an external memory mechanism to the navigation policy to leverage past experiences. To this end, we first train a state-embedding network in a self-supervised fashion. This network, trained to detect nearby locations, allows us to build an external memory containing representations of past observations. To leverage this memory, we add an attention module to the navigation policy so it can condition on this memory.

4.3.1 Training a State-Embedding Network

Before learning the navigation policy, we train a state-embedding network to learn representations of the environment's locations. The motivation for introducing this network is to encourage nearby locations in the environment to have similar representations, while pushing distant locations to have different ones. However, since we do not have access to the agent's position, the notion of distance between locations in the environment cannot be computed directly. As in [28], we will use the number of steps taken by an agent with a random policy to approximate the distance between locations.

We let an agent with random policy explore the environment for T steps and denote by $(x_1, ..., x_T)$ the corresponding sequence of observations. We ensure that the length of the computed sequence is T by removing the `STOP` action from the action space. We then define a reachability label y_{ij} for each pair of observations (x_i, x_j) that depends on their distance in the sequence. More precisely,

$$y_{ij} = \begin{cases} 1 & \text{if} \quad |i-j| \leq k \\ 0 & \text{otherwise} \end{cases} \qquad \text{for } 1 \leq i, j \leq T \tag{3}$$

where k is a hyperparameter.

We train a siamese neural network R, the Reachability Network [28], to predict the reachability label y_{ij} from a pair of observations (x_i, x_j). R is defined by a convolutional network g to embed the observations and a fully-connected network f to compare the embeddings. Therefore, we have

$$R(x_i, x_j) = f(g(x_i), g(x_j)). \tag{4}$$

Note that this stage of training is unsupervised since no reward signal is needed. Also, we apply the same data augmentation techniques to observations during this Reachability Network training.

4.3.2 Episodic and Long-term Memories

Once we have a state-embedding network that can differentiate observations from nearby and distant locations, the agent can collect a compact memory of its previously visited states. It will then have two distinct memory types: an episodic memory that contains elements from the current episode only, and a long-term memory that gathers information across multiple episodes.

Episodic Memory. We follow a process similar to Savinov *et al.* [28] for building episodic memory. At timestep t, the agent has a memory buffer M_{t-1} with embeddings from observations seen at previous timesteps. Since storing every observation seen by the agent would be inefficient, we store only observations that are considered novel, i.e., distant from the current memory vectors. In other words, at each timestep, we use the network R to compute a reachability score between the current observation x_t and the memory buffer M_{t-1} such that

$$r(x_t, M_{t-1}) = \max\{\, f(g(x_t), m) \mid m \in M_{t-1}\}. \quad (5)$$

We then use the following rule to update the memory buffer:

$$M_t = \begin{cases} M_{t-1} \cup g(x_t) & \text{if } r(x_t, M_{t-1}) < \tau, \\ M_{t-1} & \text{otherwise,} \end{cases} \quad (6)$$

where τ is the *reachability threshold* hyperparameter. The episodic memory is reset after each episode.

Long-Term Memory. In contrast to the episodic memory that contains only information about the current episode, long-term memory includes information from previous episodes. Given a scene s in the dataset, we denote by L^s_t the state of the long-term memory at timestep t in scene s. This memory is conserved over consecutive episodes, and is reset after p episodes in the same scene. Within an episode, the update rule is similar to that of episodic memory:

$$L^s_t = \begin{cases} L^s_{t-1} \cup g(x_t) & \text{if } r(x_t, L^s_{t-1}) < \tau, \\ L^s_{t-1} & \text{otherwise.} \end{cases} \quad (7)$$

4.3.3 Attention Module

The navigation policy can use both the episodic and long-term memories to move towards the target direction. Rather than using an explicit planner on the memories, we take advantage of the information stored in them implicitly, using an *attention module*.

The attention module has a multi-layer architecture similar to Transformers [31]. Each layer consists of a multi-head attention sublayer (`Attn`), followed by a feedforward sublayer (`FF`). See [31] for more details about these sublayers. Unlike Transformers, however, our attention module attends over a fixed set of vectors. In particular, the first L layers attend to the episodic memory, while the last L layers attend to the long-term memory:

$$\begin{aligned} z^l_t &= \texttt{FF}\left(\texttt{Attn}(z^{l-1}_t, M_t)\right) \quad \text{for } l \le L, \\ z^l_t &= \texttt{FF}\left(\texttt{Attn}(z^{l-1}_t, L^s_t)\right) \quad \text{for } L < l \le 2L. \end{aligned} \quad (8)$$

Here, z^l_t is the output from the l-th layer, but the initial input z^0_t is obtained by a linear transformation of the joint representation computed in Eq. (1): $z^0_t = W w^{\text{joint}}_t$.

The output from the attention module is then concatenated with the joint representation in Eq. (2), so the final action is now computed by

$$\pi(a_t | x_t, x^*) = \texttt{FC}(\texttt{cat}(w^{\text{joint}}_t, z^{2L}_t)). \quad (9)$$

5. Experimental Results

5.1. Implementation Details

Task Setup. We conducted all of our experiments on the Habitat simulator with the Gibson [34] dataset, which contains a set of visually-realistic indoor scenes. We use the standard 72/14 train/test scene split for this dataset. As stated above, we do not use the agent's pose or depth sensor information. The forward step range and turn angle is set to $(0.25\text{m}, 10°)$ for navigation episodes and $(1\text{m}, 30°)$ when training the reachability network. The maximum number of steps in an episode is 1k, and the success distance l is 1m.

Training the Reachability Network. We generate one trajectory per train scene from an agent with a random policy. We allow 5k steps for each trajectory and remove the stop action from the action space. This procedure results in a total of 360k steps from 72 scenes to train the reachability network. From each trajectory, we sample 1k positive pairs (within $k = 10$ timesteps) and 1k negative pairs, yielding a dataset of 144k image pairs. We implement the reachability network as a siamese network with a ResNet18 for the g function (see Section 4 for notations). Each of the v views from the RGB observation is passed through the ResNet separately. We sum the resulting outputs to form the embedding vector of a panoramic observation. The comparison function f is composed of two hidden layers of dimension 512. We train this network using SGD for 30 epochs with a batch size of 256, a learning rate of 0.01, a momentum of 0.9, a weight decay of 10^{-7}, and no dropout.

Training Data for the Navigation Policy. We generated 9k navigation episodes in each train scene. Following the protocol from [9], we split our navigation episodes into three levels of difficulty, based on the distance between the start and goal locations: *easy* (1.5 - 3m), *medium* (3 - 5m), and *hard* (5 - 10m). For each scene, we sample 3k start/goal location pairs per level of difficulty. Similarly, we sample 100 episodes per test scene and per level of difficulty, resulting in 4.2k test episodes. The exact episodes used in this work will be made available as a dataset.

Adding Training Scenes. In some of our experiments, to improve the generalization of our model, we used an extended set of scenes from the Gibson dataset. Appart from the 72 standard training scenes used for comparison, we

Model	Setup	Easy		Medium		Hard		Overall	
		Succ	SPL	Succ	SPL	Succ	SPL	Succ	SPL
ResNet + GRU + IL [9]		0.57	0.23	0.14	0.06	0.04	0.02	0.25	0.10
Target-Driven RL [37]		0.56	0.22	0.17	0.06	0.06	0.02	0.26	0.10
Active Neural SLAM [6]		0.63	0.45	0.31	0.18	0.12	0.07	0.35	0.23
Neural Topological SLAM [9]		**0.80**	**0.60**	0.47	0.31	0.37	0.22	0.55	0.38
Our Method	✓ Comparable	0.69	0.53	**0.62**	**0.48**	**0.54**	**0.40**	**0.61**	**0.47**
Our Method	✗ Extended	0.77	0.60	0.72	0.56	0.64	0.47	0.71	0.54

Table 1. Comparison of our proposed model with several baselines and recent approaches. We compare two versions of our method: the first one trained on the standard dataset and without long-term memory is comparable to other methods, while the second one is trained on an extended set of scenes and uses long-term memory. We report success rate and SPL, over three levels of difficulty. Our method significantly improves over all the previous approaches, almost doubling the SPL on *hard* episodes.

consider 150 additional scans which are usually rated as being of poor quality. Using this additional data, we generated an extended training set by sampling 9k navigation episodes from 222 scenes in total.

Navigation Policy Implementation. At the beginning of each episode, the simulator generates the observation from the goal location as a $v \times 3 \times 128 \times 128$ panoramic RGB image and gives it to the agent as target observation. We use a ResNet18 with shared weights for encoding the current and target observations in the policy. The size of the embedding space is 512. We concatenate the encoder's outputs for the v views of the current observation and feed into an LSTM with two recurrent layers. Our attention module consists of 4 stacked layers of a 4-headed attention network. We set the buffer's capacity to 20 for the episodic memory and 100 for the long-term memory. We train the policy using DDPPO for 50k updates, with 2 PPO epochs, a forward of 64 steps, an entropy coefficient of 0.01, and a clipping of 0.2. We use the Adam optimizer with a learning rate of $2.5\text{e}-4$.

Data Augmentation. For the training stages of both the reachability network and the navigation policy, we used random cropping with a minimum scale of 0.8 and color jittering with value 0.2 for brightness, contrast, saturation, and hue levels. These transformations are applied at two different levels when training the navigation policy: (i) when the agent samples the action a_t from the policy, and (ii) during the forward-backward in PPO. Note that, for these two steps, the transformation applied to the images is independent and results in different input images.

Baselines. To compare with previous works, we follow the protocol of [9]. We compared the performance of our model against the methods that are reported in this work: **ResNet + GRU + IL** [9], **Target-Driven RL** [37], **Active Neural SLAM** [6] and **Neural Topological SLAM** [9].

5.2. Image-Goal navigation

As a first empirical evaluation, we compare our method against the aforementioned state of the art and baselines on the image-goal navigation task. We compare with two versions of our method: the first one was trained on the standard training set and does not use the long-term memory. It is therefore fully-comparable to baselines as it does not use extra information. The second one was trained on the extended training set (as described in Section 5.1) and uses the long-term memory. It therefore uses privileged information and is not directly comparable to other methods. We trained both our models for 500M steps for three random seeds, and evaluated on the corresponding validation set. For the baselines, we report numbers from [9]. We show the results of this experiment in Table 1.

The performance obtained with the first version of our memory-augmented policy is superior to that of previous work by a significant margin ($+9\%$ SPL on average). We obtain this strong performance while - as opposed to previous work - not using any kind of position information. What is interesting is that the improvement over [9] is the strongest on harder episodes. While the proposed approach fails to improve on *easy* episodes, the improvement is of $+17\%$ SPL for *medium* and $+18\%$ for *hard*. This is particularly interesting as *medium* and *hard* constitute actual navigation tasks with the foal not being in direct line of sight from the start. The poor performance on *easy*, is potentially due to the lack of position features or using a discretized panoramic observation. In [9], the authors use equally divided equirectangular projection of the surroundings.

For the second version of our method with long-term memory and trained on an extended set of scenes, the improvement over previous methods is even bigger: $+25\%$ SPL for *medium*, $+26\%$ for *hard*, and $+18\%$ on average.

Steps	Model	Easy	Med.	Hard	Overall
250M	RL	0.41	0.35	0.24	0.33
	+ Data Aug.	0.48	0.43	0.35	0.42
	+ Ep. Memory	0.49	0.45	0.37	0.44
	+ LT Memory	0.50	0.45	0.38	0.45
	+ More scenes	0.53	0.47	0.40	0.47
500M	RL	0.44	0.37	0.26	0.36
	+ Data Aug.	0.55	0.50	0.42	0.49
	+ Ep. Memory	0.53	0.48	0.40	0.47
	+ LT Memory	0.53	0.49	0.41	0.48
	+ More Scenes	0.60	0.56	0.47	0.54

Table 2. Ablation study of our method. We present the SPL obtained with five variants of the model. For completeness, we report the performance of models trained for 250M simulation steps as well as for 500M steps.

5.3. Ablation Study and Analysis

In order to understand the design choices in our model better, we conduct several analysis experiments. We evaluate the impact of panoramic observations, discuss the regularization effect of data augmentation and do an ablation study on the setup presented in Table 1.

Ablation Study. We carry out an ablation study, in which we evaluate the performance of our basic target-driven RL, as well as the improvements brought by data augmentation, the memory-based model and adding more scenes to the training set. To this end, we train five variants of our model: (i) **RL**, the vanilla target-driven RL baseline to which we consecutively add (ii) **Data Aug.**, the data augmentation module, (iii) **Ep. Memory**, the episodic memory module, (iv) **LT Memory**, the long-term memory module, and (v) **More Scenes**, the whole model trained on the extended set of scenes. We train these models for 250M and 500M steps for three random seeds and report the average SPL obtained by the agent on the test set, on *easy*, *medium* and *hard* episodes. We present the results in Table 2.

First of all, we see that using data augmentation when training a RL-based navigation policy in this context improves the SPL significantly. When trained for 250M steps, the gap with the vanilla baseline is $+9\%$ overall and grows to $+13\%$ when the model is trained further. The gains from augmenting the training data with colorjitter and crops seems to be more important on *hard* episodes ($+16\%$) than *easy* ones ($+11\%$). Second, we observe that the long-term memory-based policy improves over the extremely competitive data-augmented baseline when trained for 250M steps ($+3\%$ on average). When trained further, the gap then shrinks and the proposed memory-based model works slightly worse than the data-augmented one (-1%). We posit that this is partly due to the limited definition of the navigation task. Indeed, the proposed model is designed to help navigation in complex episodes, where the goal is further away. In order to test this hypothesis, we further evaluate this aspect in Sec. 5.4. Finally, we observe that adding more training scenes improves significantly the generalization of the model. The performance gap with the model trained on the standard dataset is $+2\%$ SPL after 250M steps and grows to $+6\%$ when the model is trained further.

Number of views	1	3	4	6
SPL	0.08	0.31	0.36	0.36
Frames per sec.	1890	2000	2080	2340

Table 3. Analysis of the SPL obtained with the RL baseline for various panoramic view configurations. We report the average SPL as well as the number of frames that we can process per second for the given configuration.

Panoramic Observations. As a first analysis, we study the impact of the nature of panoramic observations on the final navigation performance. As described in Sec. 3, we generate panoramic observations by equally spaced planar observations around the agent. In this experiment, we compare the performance of a vanilla Target-Driven RL model trained with 1, 3, 4 and 6 views around the agent. We let the model train for 500M steps for three random seeds and report the average SPL obtained by this agent in Table 3.

The first thing we observe from Table 3, is that an agent trained with a single view i.e. without panoramic observations, completely fails to learn a successful policy, obtaining only 0.08 SPL. This result is quite intuitive, as the relative localization with respect to the goal is made much easier by multiple views, removing the need for additional search if the goal is behind the agent. Better performance is obtained with either four or six views, with an SPL of 0.36. Second, we see that there is a tradeoff between the performance we obtain and the additional runtime required with more views. Because of that, in our experiments, we run all the variants of the models with four views.

Data Augmentation and Overfitting. Here, we evaluate the capacity of our navigation agent to generalize to unseen environments. We investigate how data augmentation allows us to bridge the train / test gap observed on this task. To this end, we plot the train and test SPL during the training phase of the agent. We report this experiment for the vanilla Target-Driven RL, as well as for the same model with data augmentation in Fig. 4.

By looking at the train and test SPL for the two methods, we see that the generalization gap is huge (almost 65% for the RL baseline). The use of data augmentation allows to remedy this problem reducing this gap to about 40%. An interesting observation is that data augmentation not only improves the test performance but also helps faster conver-

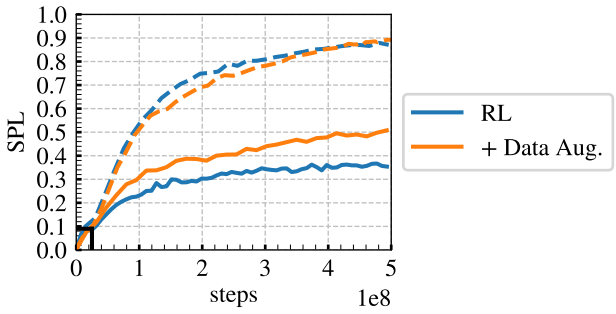


Figure 4. Performance measured in SPL as a function of training steps taken in the environment for the RL baseline (RL) and with data augmentation (+ Data Aug.). We report both the training SPL (dashed line) and the test SPL (solid line) on the same figure. The generalization gap is large and can be reduced by using classical computer vision data augmentation. The black mark at 25M steps correspond to the **Target-Driven RL** baseline as reported in [9].

Model	Easy	Med.	Hard	Extra	Overall
RL	0.29	0.24	0.17	0.04	0.18
+ Data Aug.	0.36	0.32	0.27	**0.11**	0.27
+ Ep. Memory	0.38	0.37	0.32	**0.11**	0.29
+ LT Memory	**0.42**	**0.40**	**0.35**	**0.11**	**0.32**

Table 4. Evaluating our model and the ablated variants on the **extra dataset**. We train our model for 500M steps on a combination of *easy, medium*, *hard*, *extra* episodes. The resulting dataset contains on average much longer training episodes, making the learning harder.

gence ($+2\%$ on the train SPL), as opposed to what is usually observed in traditional supervised scenarios.

Finally, we set a mark on the train and test SPL at 25M steps, which is the number of environment steps used to train the **Target-Driven RL** baseline reported in [9]. We see that the performance of a reinforcement-based model with so few steps is quite poor. Models trained with RL are really trained on-line, blurring the line between number of samples and number of model updates. At the same time, models such as [6, 9], trained with supervised learning objectives require a lower number of samples, but still need multiple passes over the data to train properly.

5.4. Training on Extra-Hard Data

Following the ablation study carried out in Sec. 5.3, we observe that when trained for longer, the improvement brought by our model over the data-augmented baseline shrinks. We posit that this is because the original dataset composed of *easy*, *medium* and *hard* episodes is too simple. We generated an additional portion of training and test episodes that we refer to as *extra*. These episodes have a goal that is between 10 and 15m away from the start location, thus are more challenging. We train two versions of our model and two baselines on *easy*, *medium*, *hard*, and *extra* episodes and report results in Table 4.

First, when the model is trained on the extended training set, the performance on the test set is slightly worse than when trained on the standard training set. This is due to the fact that, when adding the *extra* split to the training set, the agent spends less time learning to navigate on *easy*, *medium* and *hard* episodes. Second, we observe that while our memory module did not lead to better performance than the baseline with data augmentation on the standard set, it actually outperforms this baseline when trained on the extended set. The improvement over data-augmentated RL is $+8\%$ SPL on *medium* episodes and $+9\%$ SPL on *hard* ones.

5.5. Qualitative Results

Figure 5 shows a few success and failure cases from episodes of the test dataset. From the success cases, we see that our agent successfully learnt to navigate to challenging locations, that are distant from the start location (5-a) and/or located at extremities of the scenes (5-c, 5-d). Moreover our agent shows interesting skills, like bypassing obstacles (5-c) or look around in a room (5-b). From the failure cases, we see that our agent has still some undesired behaviour. For example, it can get stuck in loops (5-f), stop too early (5-h), or fail to reach some extremely challenging goals (5-g).

6. Conclusion

In this paper, we have presented a memory-endowed agent that we propose to train with reinforcement learning. This memory is accessed in the navigation policy using a transformer-inspired neural network, with attention modules over past experiences. We leverage classical computer vision data augmentation strategies to regularize the learning of the model. We evaluated our agent on the challenging task of image-goal navigation, and have shown that it surpasses previous work by a fair margin. When we train our model without the proposed memory module, it constitutes a very strong RL-based baseline for the task. In order to showcase the performance of the agent in a very complex setup, we extended the dataset proposed by [9] by adding episodes with goals 10-15m away. In future work, we plan to improve the training of the state-embedding network and allow the policy to better exploit information in the memory.

Acknowledgements. Karteek Alahari is supported in part by the ANR grant AVENUE (ANR-18-CE23-0011).

References

[1] Gil Avraham, Yan Zuo, Thanuja Dharmasiri, and Tom Drummond. Empnet: Neural localisation and mapping using embedded memory points. In *Int. Conf. Comput. Vis.*, 2019. 2

[2] Dhruv Batra, Aaron Gokaslan, Aniruddha Kembhavi, Oleksandr Maksymets, Roozbeh Mottaghi, Manolis Savva,

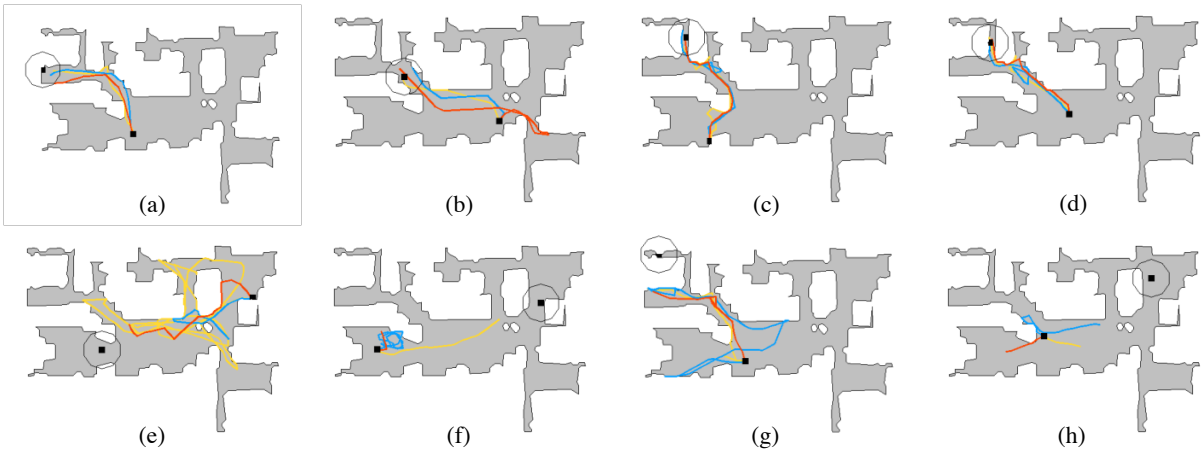


Figure 5. Qualitative Results. The top (resp. bottom) row shows trajectories from test episodes with highest (resp. lowest) SPL on the Eastville scene. Start and goal locations are shown in black, with the goal being circled by a line showing the success area. The results are shown for 3 seeds on the *hard* validation split, for our model with memory trained on the standard dataset.

Alexander Toshev, and Erik Wijmans. Objectnav revisited: On evaluation of embodied agents navigating to objects. *arXiv preprint arXiv:2006.13171*, 2020. 1

[3] Edward Beeching, Jilles Dibangoye, Olivier Simonin, and Christian Wolf. Learning to plan with uncertain topological maps. In *Proc. ECCV*, 2020. 2

[4] Edward Beeching, Christian Wolf, Jilles Dibangoye, and Olivier Simonin. EgoMap: Projective mapping and structured egocentric memory for deep RL. *arXiv preprint arXiv:2002.02286*, 2020. 2

[5] Marvin Chancán and Michael Milford. MVP: Unified motion and visual self-supervised learning for large-scale robotic navigation. *arXiv preprint arXiv:2003.00667*, 2020. 2

[6] Devendra Singh Chaplot, Dhiraj Gandhi, Saurabh Gupta, Abhinav Gupta, and Ruslan Salakhutdinov. Learning to explore using active neural SLAM. *arXiv preprint arXiv:2004.05155*, 2020. 3, 6, 8

[7] Devendra Singh Chaplot, Helen Jiang, Saurabh Gupta, and Abhinav Gupta. Semantic curiosity for active visual learning. *arXiv preprint arXiv:2006.09367*, 2020. 3

[8] Devendra Singh Chaplot, Emilio Parisotto, and Ruslan Salakhutdinov. Active neural localization. *arXiv preprint arXiv:1801.08214*, 2018. 2

[9] Devendra Singh Chaplot, Ruslan Salakhutdinov, Abhinav Gupta, and Saurabh Gupta. Neural topological SLAM for visual navigation. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2020. 2, 5, 6, 8

[10] Kevin Chen, Juan Pablo de Vicente, Gabriel Sepulveda, Fei Xia, Alvaro Soto, Marynel Vázquez, and Silvio Savarese. A behavioral approach to visual navigation with graph localization networks. *arXiv preprint arXiv:1903.00445*, 2019. 2

[11] Tao Chen, Saurabh Gupta, and Abhinav Gupta. Learning exploration policies for navigation. *arXiv preprint arXiv:1903.01959*, 2019. 3

[12] Howie Choset and Keiji Nagatani. Topological simultaneous localization and mapping (SLAM): Toward exact localization without explicit localization. *IEEE Transactions on robotics and automation*, 17(2):125–137, 2001. 2

[13] Abhishek Das, Samyak Datta, Georgia Gkioxari, Stefan Lee, Devi Parikh, and Dhruv Batra. Embodied question answering. In *IEEE Conf. Comput. Vis. Pattern Recog. Worksh.*, 2018. 1

[14] Ben Eysenbach, Russ R Salakhutdinov, and Sergey Levine. Search on the replay buffer: Bridging planning and reinforcement learning. In *Advances in Neural Information Processing Systems*, 2019. 2

[15] Kuan Fang, Alexander Toshev, Li Fei-Fei, and Silvio Savarese. Scene memory transformer for embodied agents in long-horizon tasks. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2019. 2

[16] Saurabh Gupta, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive mapping and planning for visual navigation. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2017. 2

[17] Dinesh Jayaraman and Kristen Grauman. Learning to look around: Intelligently exploring unseen environments for unknown tasks. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2018. 3

[18] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *arXiv preprint arXiv:1806.10293*, 2018. 2

[19] Ilya Kostrikov, Denis Yarats, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. *arXiv preprint arXiv:2004.13649*, 2020. 2

[20] Ashish Kumar, Saurabh Gupta, David Fouhey, Sergey Levine, and Jitendra Malik. Visual memory for robust path

following. In *Advances in Neural Information Processing Systems*, 2018. 2

[21] Michael Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. Reinforcement learning with augmented data. *arXiv preprint arXiv:2004.14990*, 2020. 2

[22] Lina Mezghani, Sainbayar Sukhbaatar, Arthur Szlam, Armand Joulin, and Piotr Bojanowski. Learning to visually navigate in photorealistic environments without any supervision. *arXiv preprint arXiv:2004.04954*, 2020. 2, 3

[23] Piotr Mirowski, Matt Grimes, Mateusz Malinowski, Karl Moritz Hermann, Keith Anderson, Denis Teplyashin, Karen Simonyan, Andrew Zisserman, Raia Hadsell, et al. Learning to navigate in cities without a map. In *Advances in Neural Information Processing Systems*, 2018. 2

[24] Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andrew J Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, et al. Learning to navigate in complex environments. *arXiv preprint arXiv:1611.03673*, 2016. 2

[25] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. ORB-SLAM: A versatile and accurate monocular slam system. *IEEE transactions on robotics*, 31(5):1147–1163, 2015. 2

[26] Emilio Parisotto and Ruslan Salakhutdinov. Neural map: Structured memory for deep reinforcement learning. *arXiv preprint arXiv:1702.08360*, 2017. 2

[27] Nikolay Savinov, Alexey Dosovitskiy, and Vladlen Koltun. Semi-parametric topological memory for navigation. *arXiv preprint arXiv:1803.00653*, 2018. 2

[28] Nikolay Savinov, Anton Raichuk, Raphaël Marinier, Damien Vincent, Marc Pollefeys, Timothy Lillicrap, and Sylvain Gelly. Episodic curiosity through reachability. *arXiv preprint arXiv:1810.02274*, 2018. 3, 4, 5, 11

[29] Sebastian Thrun. Probabilistic robotics. *Communications of the ACM*, 45(3):52–57, 2002. 2

[30] Nicola Tomatis, Illah Nourbakhsh, and Roland Siegwart. Combining topological and metric: A natural integration for simultaneous localization and map building. In *Proc. European Workshop on Advanced Mobile Robots (Eurobot)*. ETH-Zürich, 2001. 2

[31] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, 2017. 2, 5

[32] Erik Wijmans, Abhishek Kadian, Ari Morcos, Stefan Lee, Irfan Essa, Devi Parikh, Manolis Savva, and Dhruv Batra. DD-PPO: Learning near-perfect pointgoal navigators from 2.5 billion frames. *arXiv preprint arXiv:1911.00357*, 2019. 1, 2

[33] Yi Wu, Yuxin Wu, Aviv Tamar, Stuart Russell, Georgia Gkioxari, and Yuandong Tian. Bayesian relational memory for semantic visual navigation. In *Int. Conf. Comput. Vis.*, 2019. 2

[34] Fei Xia, Amir R Zamir, Zhiyang He, Alexander Sax, Jitendra Malik, and Silvio Savarese. Gibson env: Real-world perception for embodied agents. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2018. 2, 5

[35] Wei Yang, Xiaolong Wang, Ali Farhadi, Abhinav Gupta, and Roozbeh Mottaghi. Visual semantic navigation using scene priors. *arXiv preprint arXiv:1810.06543*, 2018. 2

[36] Jingwei Zhang, Lei Tai, Joschka Boedecker, Wolfram Burgard, and Ming Liu. Neural SLAM: Learning to explore with external memory. *arXiv preprint arXiv:1706.09520*, 2017. 2

[37] 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 *Proc. ICRA*, 2017. 1, 2, 6

A. Analysis of the Reachability Network

We begin with additional details about training the reachability network and then present a few qualitative visualisations.

A.1. Architecure & Training Details

We adapted the architecture of the reachability network presented in [28] to work with panoramic views in realistic environments. An illustration of this model is shown in Figure 6. The main differences with [28] are the use of a data augmentation module, that applies transformations to the RGB input of the network, and an additional layer to handle panoramic observations. This layer aggregates the ResNet output for each of the v views into one feature vector. We experimented with two architectures for this layer: (i) **cat + FC**: where we concatenate the ResNet output for each of the v views and feed this large vector into a one-layer fully-connected module, and (ii) **sum**: where we simply sum these v vectors. Contrary to (i), (ii) has the interesting property of yielding embeddings that are rotation invariant—the resulting feature vector from a location of the environment will be the same for every direction of the agent.

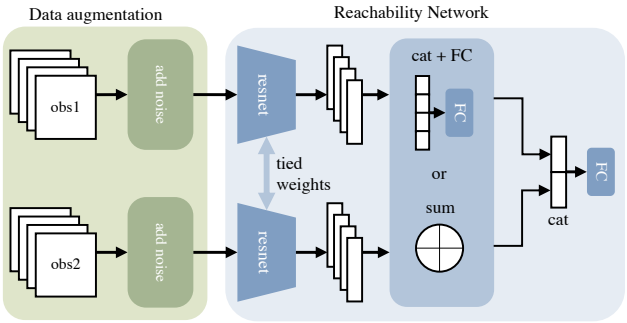


Figure 6. Architecture of the reachability network. We adapted the architecure from [28] by using a data augmentation module and a layer that handles panoramic observations. The output of the last fully-connected module is the similarity score between the two observations.

We compare the train and validation performance of these two design variants, and the influence of using data augmentation for training the reachability network. The results are shown in Figure 7. First, we observe that using data augmentation reduces overfitting in both the setups, and yields a better validation accuracy in these two cases. Second, we see that summing panoramic features allows to achieve better train and validation performance than concatenating them. One explanation for this is that the rotation invariance explained above is facilitating the learning. For all of our navigation policy learning experiments, we chose the setup **sum + augmentation** for training the reachability networks that encodes the memory vectors.

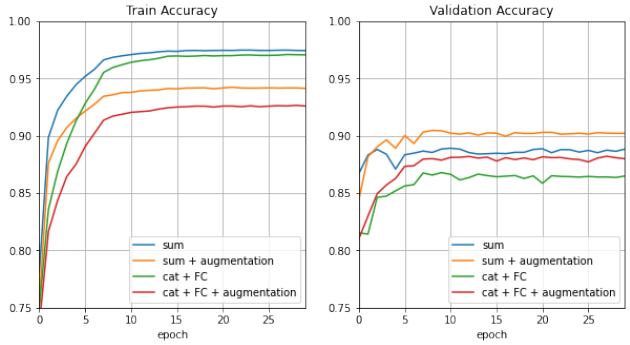
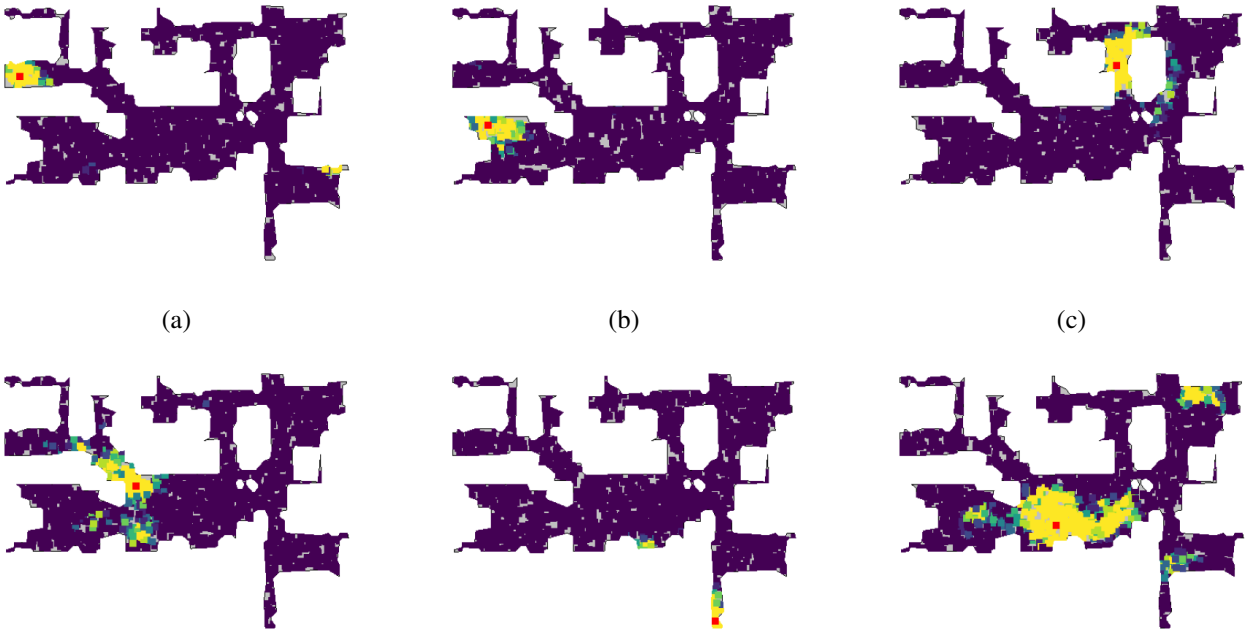


Figure 7. Training and validation curves for the reachability network. We tested four setups, that compare the use of data augmentation and the choice of the aggregation layer for panoramic features.

A.2. Qualitative Visualisations

We visualise the quality of the reachability network with the following experiment. First, we put the agent at a random location in the environment and sample an observation x from there. Then, we randomly sample N observations in the environment and for each of these observations, we compute their similarity score with observation x, using the reachability network. We present these results on a heat-map, where the colour at a location represents the corresponding similarity score. Some examples are shown in Figure 8. We see that the high similarity scores are at locations that are around the comparison observation, which implies that the reachability network performs well at learning representations that are similar for nearby locations, and dissimilar for representations that are far away. Since these experiments are shown on a validation environment, we note that the reachability network generalizes well to unseen environments.

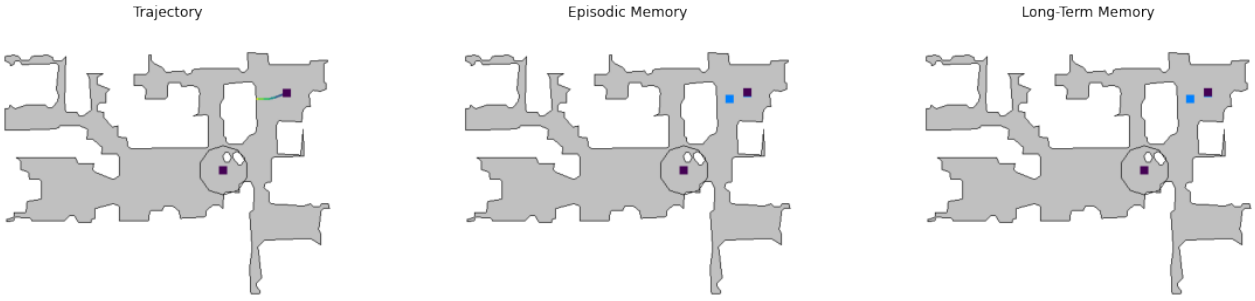
We also visualise the state of the episodic and long-term memories for consecutive validation episodes in Figure 9. From this, we observe how these memories are filled through consecutive validation episodes. After 100 navigation episodes (9-d), the long-term memory is well filled and covers most of the environment. This allows the agent to reach challenging goals.



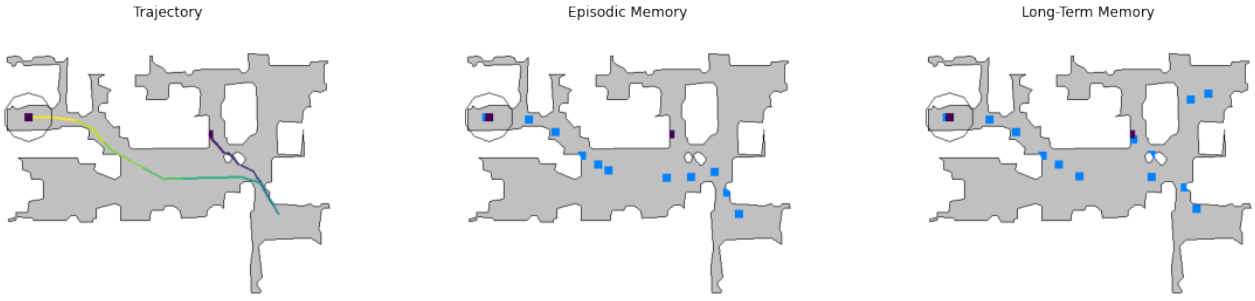
(a) (b) (c)

(d) (e) (f)

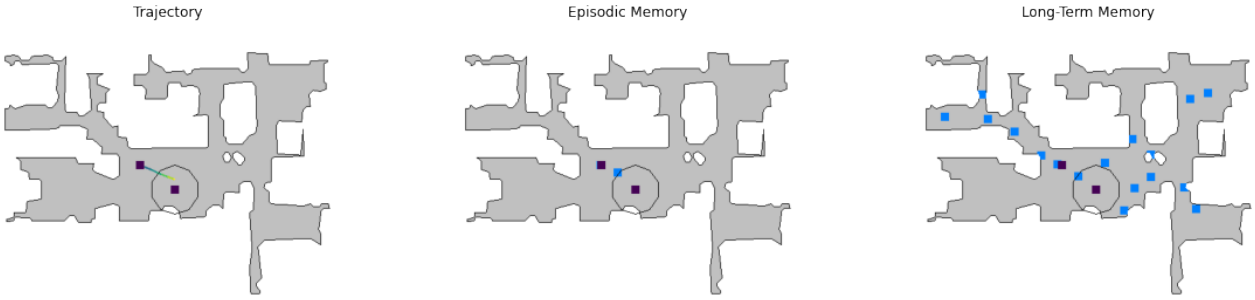
Figure 8. Heat-maps of the similarity score between the observation (the red point) and the observations at $N = 2000$ points sampled randomly in the environment. The colour at a location corresponds to the similarity score at that location: low values, close to 0, are in dark violet and high values, close to 1, in yellow. These visualisations were performed on Eastville: an environment of the validation set.



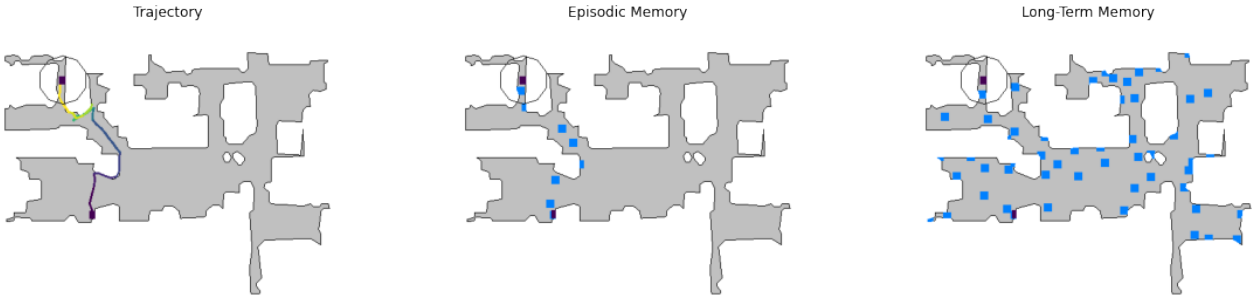
(a) Step 1



(b) Step 2



(c) Step 5



(d) Step 100

Figure 9. Visualisation of the agent's trajectory, episodic and long-term memories for first, second, fifth and 100th episode in the Eastville environment. The start and goal locations are shown in black, goal location being circled by a line showing the success area. The blue points represent the location of the episodic and long-term memory vectors. The episodic memory is reset after each episode, while the long-term memory remains for 100 episodes in the same scene.