Neural SLAM: Learning to Explore with External Memory

Jingwei Zhang[*]   **Lei Tai**[†]   **Joschka Boedecker**[*]   **Wolfram Burgard**[*]   **Ming Liu**[†]

Abstract

We present an approach for agents to learn representations of a global map from sensor data, to aid their exploration in new environments. To achieve this, we embed procedures mimicking that of traditional simultaneous localization and mapping (SLAM) into the soft attention based addressing of external memory architectures, in which the external memory acts as an internal representation of the environment for the agent. This structure encourages the evolution of SLAM-like behaviors inside a completely differentiable deep neural network. We show that this approach can help reinforcement learning agents to successfully explore new environments where long-term memory is essential. We validate our approach in both challenging grid-world environments and preliminary Gazebo experiments. A video of our experiments can be found at: `https://goo.gl/G2Vu5y`.

1 Introduction

1.1 Cognitive Mapping

Studies of animal navigation have shown that the hippocampus plays an important role (O'keefe and Nadel 1978) (McNaughton et al. 2006) (Collett and Graham 2004). It performs cognitive mapping that combines path integration and visual landmarks, so as to give the animals sophisticated navigation capabilities instead of just reflexive behaviors based only upon the immediate information they perceive.

Similarly, to successfully navigate and explore new environments in a timely fashion, intelligent agents would benefit from having their own internal representation of the environment whilst traverse, so as to go beyond the scope of performing reactive actions based on the most recent sensory input. Traditional methods in robotics thus developed a series of methods like simultaneous localization and mapping (SLAM), localization in a given map, path planning and motion control, to enable robots to complete such challenging tasks (Thrun, Burgard, and Fox 2005) (LaValle 2006) (Latombe 1991). Those individual components have been well studied and understood as separate parts, but here we

view them as a unified system and attempt to embed SLAM-like procedures into a neural network such that SLAM-like behaviors maybe be able to out of the course of reinforcement learning agents exploring new environments. This guided learned system could then benefit from each individual component (localization, mapping and planning) adapting in the awareness of each other's existence, instead of being rigidly combined together as in traditional methods. Also, in this paper we represent this system using a completely differentiable deep neural network, ensuring the learned representation is distributed and feature-rich, a property that rarely comes with traditional methods but is key to robust and adaptive systems (Bengio 2013).

1.2 External Memory

The memory structure in traditional recurrent neural networks (RNNs) like long short term memory networks (LSTMs) are ultimately short-term, which would not be sufficient for developing informative navigation or exploration strategies. For the network to have an internal representation of the environment, i.e., its own cognitive map, an external memory architecture (Graves, Wayne, and Danihelka 2014) (Graves et al. 2016) is required. Having an external memory besides a deep network separates the learning of computation algorithms from the storage of data over long time-scales. This is essential for learning successful exploration strategies, since if the computation and the memory are mixed together in the weights of the network, then with the memory demands increasing over time, the expressive capacity of the network would be very likely to decrease (Graves et al. 2016).

Besides the neural turning machine (NTM) (Graves, Wayne, and Danihelka 2014) and the differentiable neural computer (DNC) (Graves et al. 2016), there is another branch of work on external memory architectures for deep networks which studies the memory networks. But the memory networks as in (Oh et al. 2016) (Sukhbaatar et al. 2015) do not learn what to write to the memory, which is not sufficient for our task because the network is expected to learn to map onto its external memory to aid planning.

The Neural Map as proposed in (Parisotto and Salakhutdinov 2017) adapted the $1D$ external memory in (Graves, Wayne, and Danihelka 2014) to $2D$ as a form of structured map storage for an agent to learn to navigate. However, they

---

[*]Department of Computer Science, University of Freiburg, Germany; {zhang,jboedeck, burgard}@informatik.uni-freiburg.de

[†]Department of Electronic and Computer Engineering, The Hong Kong University of Science and Technology; {ltai, eelium}@ust.hk

do not utilize the $2D$ structure of this memory as all their operations can be conducted as if the memory address were a $1D$ vector. Furthermore, they assume the location of the agent is always known so as to write exactly to the corresponding location in the memory while the agent travels through the maze, a prerequisite that can rarely be met in real-world scenarios.

1.3 Embedding Classic Models into Deep Neural Networks

Embedding domain-specific structures into neural networks can be found in many works. Unlike methods that treat the networks completely as black-box approximators thus cannot benefit from the valuable prior knowledge accumulated over the years (it is like forcing a boy to deduct all the laws of physics from his observations by himself but not giving him the physics textbook to learn from), this line of formulation biases deep models toward learning representations containing the structures that we already know they would benefit from having for specific domains.

(Tamar et al. 2016) embedded the value iteration procedures into a single network, forcing the network to learn representations following the well-defined policy-evaluation, policy-improvement loop, while benefiting from the feature-rich representations from deep architectures. (Gupta et al. 2017) went one step further by using the Value Iteration Network as the planning module inside a visual navigation system. They treat an internal part of the network as an egocentric map and apply motion on it. (Fischer et al. 2015) added a cross-correlation layer to compute correlations of features of corresponding neighboring cells between subsequent frames, which explicitly provides the network with matching capabilities. This greatly helps the learning of optical flow since the optical flow is computed based on local pixel dynamics. (Zhang et al. 2016) forced the network to learn representative features across tasks by explicitly embedding structures mimicking the computation procedures of successor feature reinforcement learning into the network, and their resulting architecture is able to transfer navigation policies across similar environments.

Traditionally, when using well-established models in a combined system with other modules, they typically do not benefit from the other components. This is because their behaviors can not adapt accordingly, as those models come out of deduction but have not evolved out of learning (directly applying those well established traditional models is like to directly give the boy all the answers to his physics questions instead of giving him the physics textbook for him to learn to solve those questions). While if those functionalities are learned along with other components, their behaviors can influence each other and the system could potentially obtain performance beyond directly combining well-established models.

Let us take SLAM as an example. SLAM is used as a building block for complicated autonomous systems to aid navigation and exploration, yet the SLAM model and the path planning algorithms are individually developed, not taking each other into account. (Bhatti et al. 2016) augmented the state space of their reinforcement learning agent with the output of a traditional SLAM algorithm. Although this improves the navigation performance of the agent, it still experiences the issues discussed above since SLAM is rigidly combined into their architecture. While if SLAM-like behaviors can be encouraged to evolve out of the process of agents learning to navigate or to explore, then the resulting system would be much more deeply integrated as a whole, with each individual component influencing each other while benefiting from learning alongside each other. The SLAM model from the resulting system would evolve out of the need for exploration or navigation, not purely just for performing SLAM. Additionally, if learning with deep neural nets, the resulting models will be naturally feature-rich, which is rarely a property of traditional well-established models.

Although a number of works have been presented on utilizing deep reinforcement learning algorithms for autonomous navigation (Mirowski et al. 2016) (Zhu et al. 2016) (Zhang et al. 2016) (Gupta et al. 2017) (Tai, Paolo, and Liu 2017), none of them have an explicit external memory architecture to equip the agent with the capability of making long-term decisions based on an internal representation of a global map. Also, these works mainly focus on learning to navigate to a target location, while in this paper we attempt to solve a more challenging task of learning to explore new environments under a time constraint, in which an effective long term memory mechanism is essential.

Following these observations, we attempt to embed the motion prediction step and the measurement update step of SLAM into our network architecture, by utilizing the soft attention based addressing mechanism in (Graves, Wayne, and Danihelka 2014), biasing the write/read operations towards traditional SLAM procedures and treating the external memory as an internal representation of the map of the environment, and train this model using deep reinforcement learning algorithms, to encourage the evolution of SLAM-like behaviors during the course of exploration.

1.4 Exploration in Unknown Environments

Effective exploration capabilities are required of intelligent agents to perform tasks like surveillance, rescue and sample collection (Shen, Michael, and Kumar 2012). Traditional techniques for exploration includes information gain based approaches, goal assignment using coverage maps or occupancy grid maps, etc (Stachniss 2009). However, such techniques require building and maintaining accurate maps of the environment for the agent to memorize the already explored areas, in which loop closure plays an important role.

(Mirowski et al. 2016) added loop closure detection, along with depth prediction as auxiliary tasks to provide additional supervision signals when training reinforcement learning agents in environments with only sparse rewards. Specifically, the loop closure detection is trained via supervised learning by integrating the ground truth velocities of the agent, which is not accessible in real world scenarios. In this paper, we highlight the difference to (Mirowski et al. 2016) that the loop closure is learned implicitly within our model via an embedded SLAM structure. Our strategy requires less input information and depends less on ground-truth infor-
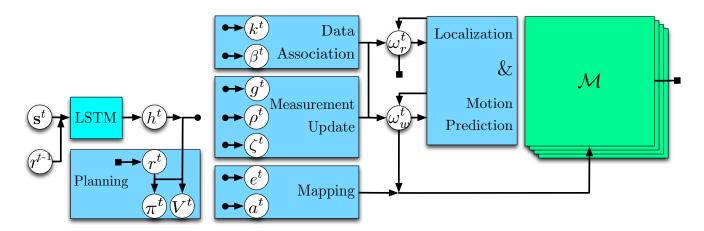
Figure 1: Visualization of the Neural-SLAM model architecture (we intentionally use *blue* for the components in charge of *computation*, *green* for *memory*, and *cyan* for a mixture of both.)

mation as supervision. Additionally, we tested our approach in the Gazebo environment (Koenig and Howard) which is more realistic with respect to the underlying physics and the sensor noise compared to the simulated environment used in (Mirowski et al. 2016). Additionally, compared with traditional SLAM-based methods, our strategy eliminates the need for building and maintaining an expensive map for each new environment and only needs a forward pass through the trained model to give out planning decisions, which runs 200Hz on CPU. This enables our agent to cope with the limited memory and processing capabilities on robotics platforms.

2 Methods

2.1 Background

We formulate the exploration task as a Markov decision process (MDP) in which the agent interacts with the environment through a sequence of observations, actions and rewards. At each time step $t \in [0, T]$ the agent receives an observation z^t (in this paper the agent receives a vector of laser ranges) of the true state $\mathbf{s}^t$ of the environment. The agent then selects an action $\mathbf{a}^t$ based on a policy $\pi(\mathbf{a}^t|\mathbf{s}^t)$, which corresponds to a motion command u^t for the agent to execute. The agent then receives a reward signal R^t and transits to the next state $\mathbf{s}^{t+1}$. The goal for the agent is to maximize the expected cumulative future reward ($\gamma \in (0, 1]$ is the discount factor):

$$V^\pi(\mathbf{s}) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R^t | \mathbf{s}^0 = \mathbf{s} \right] \quad (1)$$

Recent success in deep reinforcement learning represents the value functions or the policies with deep neural networks. In this paper we utilize the asynchronous advantage actor-critic (A3C) algorithm (Mnih et al. 2016), in which both the policy and the value functions are represented by deep neural function approximators, parameterized by θ_π

and θ_V respectively (we note that θ_π and θ_V share parameters except for their output layers, parameterized by $\mathcal{W}_\pi$ and $\mathcal{W}_V$ (Sec. 2.4)). Those parameters are updated using the following gradients ($G^t = \gamma^K V(\mathbf{s}^K) + \sum_{\tau=0}^{K} \gamma^\tau R^\tau$, with K being the rollout step) (H is the entropy of the policy, λ is the coefficient of the entropy regularization term):

$$d\theta_\pi = \nabla_{\theta_\pi} \log \pi(\mathbf{a}^t|\mathbf{s}^t; \theta_\pi)(G^t - V(\mathbf{s}^t; \theta_V))$$
$$+ \lambda \nabla_{\theta_\pi} H(\pi(\mathbf{a}^t|\mathbf{s}^t; \theta_\pi)) \quad (2)$$
$$d\theta_V = \partial(G^t - V(\mathbf{s}^t; \theta_V))^2 / \partial\theta_V \quad (3)$$

2.2 Neural SLAM Architecture

As discussed previously, we require our model to have an external memory structure for the agent to utilize as an internal representation of the environment. Thus, we added an external memory chunk $\mathcal{M}$ of size $H \times W \times C$ (containing $H \times W$ memory slots, with C channels or features for each slot), which can be accessed by the network via a write head and a read head. (We note that our work can be easily extended to multiple heads for write/read, but in this paper we only investigate with one write/read head. We also observe that the number of heads can be viewed as the number of particles as in particle filter (Thrun, Burgard, and Fox 2005).)

At each time step, we feed our input directly to an LSTM cell, which gives out a hidden state h^t. This hidden state h^t is then used in each head to emit a set of control variables $\{k^t, \beta^t, g^t, \rho^t, \zeta^t\}$ (each write head additionally emits $\{e^t, a^t\}$) through a set of linear layers. The write head and the read head then each computes their access weight (ω_w^t and ω_r^t, both of size $H \times W$) based on those control variables. Then the write head would use its access weight ω_w^t along with e^t and a^t to write to the memory $\mathcal{M}^{t-1}$, while the read head would access the updated memory $\mathcal{M}^t$ with its access weight ω_r^t to output a read vector r^t. Next, h^t and r^t are concatenated together to compute the final output: a policy distribution π^t, and an estimated value V^t, which are then

used to calculate gradients to update the whole model according to Equ. 2 and Equ. 3.

The Neural-SLAM Model Architecture is shown in Fig. 1, and we will describe the operations in each component in detail in the following section.

2.3 Embedded SLAM Structure

We use the same addressing mechanism for computing the access weights of the write head ω_w^t and the read head ω_r^t, except that the read head addressing happens after the write head updates the external memory, thus it would access the memory of the current time step. Below we describe the computations in detail, where we refer to both access weights at time step t as ω^t.

Prior Belief We view the access weights of the heads as their current beliefs. We make the assumption that the initial pose of the agent is known at the beginning of each episode. Also, the sensing range of its onboard sensor is known a priori. Then we initialize the access weight ω^0 with a Gaussian kernel centering around the initial pose, filling up the whole sensing area and summing up to 1; all other areas are assigned with weight 0. The external memory is initialized as $\mathcal{M}^0 = 0$ (we discuss in more detail this choice of initialization value in Sec. 2.5).

Localization & Motion Prediction At each time step, we first do a motion prediction, by applying the motion command the agent receives from the last time step u^{t-1} onto its access weight from the last time step (M here can be any motion model):

$$\bar{\omega}^t = M(\omega^{t-1}, u^{t-1}) \tag{4}$$

Note that since we view our external memory not as an egocentric map but as a global map, we need to first localize on the access weight before the motion model can be applied. Thus, we localize by first identifying the center of mass in the current access weight matrix as the position of the agent, then choose the direction with the largest sum of weights within the corresponding sensing area as its orientation.

Data Association Each head emits a key vector k^t of length C, which is compared with each slot (x, y) in the external memory $\mathcal{M}_{(x,y)}^t$ under a similarity measure $\mathcal{S}$ (in this paper we use cosine similarity as in Equ. 6), to compute a content-based access weight ω_c^t based on the data association score (each head also outputs a key strength scalar β^t to increase or decrease the focus of attention):

$$\omega_c^t(x, y) = \frac{\exp(\beta^t \mathcal{S}(k^t, \mathcal{M}_{(x,y)}^t))}{\sum_{(i,j)} \exp(\beta^t \mathcal{S}(k^t, \mathcal{M}_{(i,j)}^t))} \tag{5}$$

$$\mathcal{S}[\mathbf{u}, \mathbf{v}] = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\| \cdot \|\mathbf{v}\|} \tag{6}$$

Measurement Update We then perform a measurement update by the following steps.

First, the content-based access weight from this time step ω_c^t and the last access weight after motion prediction $\bar{\omega}^t$ are interpolated together using an interpolation gate scalar g^t generated by each head:

$$\omega_g^t = g^t \omega_c^t + (1 - g^t) \bar{\omega}^t \tag{7}$$

Then, a shift operation is applied based on the shift kernel ρ^t emitted by each head (in this paper ρ defines a normalized distribution over a 3×3 area), to account for the noise in motion and measurement. This shift operation can be viewed as a convolution over the access weight matrices, with ρ^t being the convolution kernel:

$$\omega_\rho^t(x, y) = \sum_{i=0}^{H-1} \sum_{j=0}^{W-1} \omega_g^t(i, j) \rho^t(x - i, y - j) \tag{8}$$

Finally, the smoothing effect of the shift operation is compensated with a sharpen scalar $\zeta^t >= 1$:

$$\omega^t(x, y) = \frac{\omega_\rho^t(x, y)^{\zeta^t}}{\sum_{(i,j)} \omega_\rho^t(i, j)^{\zeta^t}} \tag{9}$$

Mapping The write head each generates two additional vectors (both contain C elements): an erase vector e^t and an add vector a^t. Along with its access weight ω_w^t, the write head accesses and updates the external memory with the following operations:

$$\tilde{\mathcal{M}}^t = \mathcal{M}^{t-1}(1 - \omega_w^t e^t) \tag{10}$$

$$\mathcal{M}^t = \tilde{\mathcal{M}}^t + \omega_w^t a^t \tag{11}$$

2.4 Planning

After the memory has been updated to $\mathcal{M}^t$, it is accessed by the read head by its access weight ω_r^t, to output a read vector r^t (which can be seen as a summary of the current internal map):

$$r^t = \sum_{(i,j)} \omega_r^t(i, j) \mathcal{M}_{(i,j)}^t \tag{12}$$

This read vector r^t is then concatenated with the hidden state h^t, and fed into two linear layers (parameterized by $\mathcal{W}_\pi$ and $\mathcal{W}_V$ respectively) to give out the policy distribution and the value estimate:

$$\pi^t(\mathbf{a}^t | \mathbf{s}^t) = \text{Softmax}(\mathcal{W}_\pi[h^t, r^t]) \tag{13}$$

$$V^t = \mathcal{W}_v[h^t, r^t] \tag{14}$$

π^t and V^t are subsequently used for calculating losses for on-policy deep reinforcement learning, as discussed in Section 2.1. An action $\mathbf{a}^t$ is then drawn from a multinomial distribution defined by π^t during training, while a greedy action is taken during evaluation and testing.
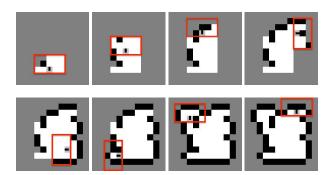
Figure 2: Visualization of a sample trajectory of a trained *Neural-SLAM* agent successfully completing exploration in a new environment. The agent is visualized as a grey grid with a black rectangle at its center pointing at its current orientation. The obstacles are shown as black grids, free space as white grids, and grey grids indicate unexplored areas. The world clears up as the agent explores with its sensor (the sensor cannot see through walls nor across sharp angles), whose sensing area is shown by red bounding boxes (the information in the red bounding box is the input to the network). An exploration is completed when the agent has cleared up all possible grids, in which case the current episode is considered to be terminated and solved. An episode would also be terminated (but not considered as solved) when a maximum step of 750 is reached.

2.5 Read-out Map from External Memory

As previously mentioned, we view the external memory $\mathcal{M}$ as an internal representation of the environment for the agent. More specifically, we treat the values stored in $\mathcal{M}$ as the log odds representation of occupancy in occupancy grid mapping techniques (Thrun, Burgard, and Fox 2005). Following this representation, we can recover the occupancy probability of all the grids (i.e., the slots on $\mathcal{M}$) following the equation below.

$$p(\mathcal{M}^t_{(x,y)}) = 1 - \frac{1}{1 + \exp\left\{\mathcal{M}^t_{(x,y)}\right\}} \qquad (15)$$

At the beginning of each episode, we set all the values in $\mathcal{M}^0$ to 0, corresponding to an occupancy probability of 0.5, to represent maximum uncertainty. We identify that Equ. 15 is identical to a sigmoid operation, thus sigmoid function is used in our implementation for this map read-out operation.

Following this formulation, one possible extension for our method would be to use $\mathcal{M}$ to compute the exploration reward R_{exp} as an internal reward signal for the agent, to eliminate the need for receiving R_{exp} from the ground truth map (for example, use the information gain from $\mathcal{M}^{t-1}$ to $\mathcal{M}^t$ as R_{exp}) (we refer to Sec. 3.1 for a detailed description of our reward structure).
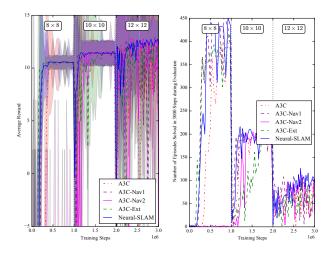

Figure 3: Comparison between the average reward obtained and number of episodes solved in 3000 steps during evaluation by an *A3C* agent (with 1 LSTM, motion command directly concatenated into the input), an *A3C-Nav1* agent (with 2 stacked LSTMs, motion command directly concatenated into the input), an *A3C-Nav2* agent (with 2 stacked LSTMs, motion command concatenated with the output of the 1st LSTM, then input into the 2st LSTM) , an *A3C-Ext* agent (with 1 LSTM and an external memory, motion command concatenated with the output of the LSTM then fed to the external memory architecture, which is like the *Neural-SLAM* without the Localization & Motion Prediction step, and our *Neural-SLAM* agent (incorportate motion command with an explicit motion model, as discussed in Sec. 2.3). We train continuously for 3 courses transitioning from world sizes of 8×8 to 12×12.

3 Experiments

3.1 Experimental Setup

We first test our algorithm in simulated grid world environments. We use a curriculum learning strategy to train agents to explore randomly generated environments ranging in size 8×8 to 12×12 (we ensure all the free grids are connected together when generating environments). At the beginning of each episode, the agent is randomly placed in a randomly generated grid world during both training and evaluation. It has a sensing area of size 3×5 (we note that this simulated laser sensor cannot see through walls nor across sharp angles) as is shown in Fig. 2.

The agent can take an action out of $\{0 : \text{Stand still}, 1 : \text{Turn Left}, 2 : \text{Turn Right}, 3 : \text{Go Straight}\}$. It receives a reward of -0.04 for each step it takes before completing the exploration task, -0.96 for colliding with obstacles, and $10.$ for the completion of an exploration. Also during the course of exploration, the agent receives a reward of $1./(3 \times 5)$ for each new grid it clears up. (We note that this requires a ground truth map, but such a map is only needed during training to provide exploration rewards, while a map is not needed during execution.)

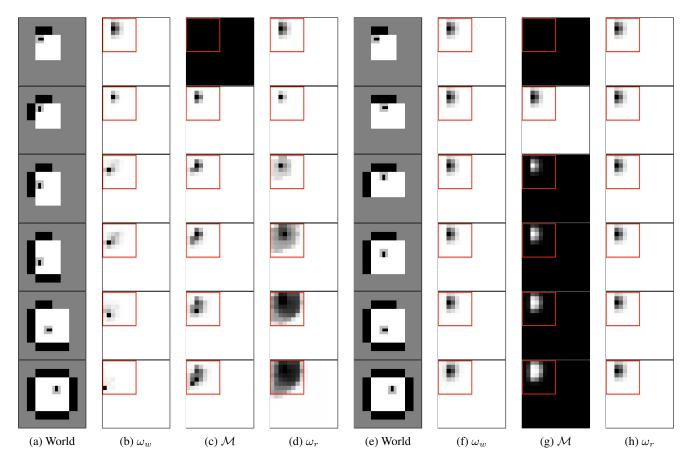At each time step, the agent receives a sensor reading of

| (a) World | (b) ω_w | (c) $\mathcal{M}$ | (d) ω_r | (e) World | (f) ω_w | (g) $\mathcal{M}$ | (h) ω_r |

Figure 4: Visualization of the world view, write head weights ω_w, memory $\mathcal{M}$ (we note that this visualization is the output of a map read-out operation (Sec. 2.5)), and read head weights ω_r during one exploration of our trained *Neural-SLAM* agent (Fig. 4a, 4b, 4c, 4d) and a trained *A3C-Ext* agent (Fig. 4e, 4f, 4g, 4h) on a simplest environment.

size 3×5, which is then fed into the network, along with the action it selected from the last time step. We train the network the same way as A3C (Mnih et al. 2016) and deploy 16 training processes purely on CPU, optimized with the ADAM optimizer (Kingma and Ba 2015) with shared statistics across all training processes, with a learning rate of $1e-4$. We also use a weight decay of $1e-4$ since we find this to be essential to stabilize training when combining external memory architectures with A3C. We set the rollout step K to be 20 and the maximum number of steps for each episode to be 750.

We experimented with four baseline agents as comparisons for our *Neural-SLAM* agent: 1) *A3C*: an A3C agent with one LSTM cell (with 128 hidden units) without external memory architectures. The action from the last time step is concatenated with the current sensor readings as the input to the network; 2) *A3C-Nav1*: an A3C agent with two stacked LSTM cells. The last action is fed into the 1st LSTM cell; 3) *A3C-Nav2*: same as *A3C-Nav1* except that the last action is fed into the 2nd LSTM cell (we note that this agent is very similar to the one proposed by (Mirowski et al. 2016), while here we only feed the selected action but not the true velocity and the reward to the 2nd LSTM since that information is

usually not available during execution); 4) *A3C-Ext*: an A3C agent with one LSTM cell, which also interacts with a $2D$ addressed external memory ($16 \times 16 \times 32$, we note that the largest map size the agents are trained on is 12×12, here we initialize the address of the external memory to be 16×16 for the generalization tests on larger map sizes, which we will discuss in Sec. 3.3), and access it using the same approach as we described in Sec. 2.3. But unlike our *Neural-SLAM* agent where the previous action is applied onto the memory through an explicit motion model, no motion prediction step (Sec. 2.3) is executed for the *A3C-Ext* agent, and the previous action is simply concatenated with the output of the LSTM and fed to the external memory structure.

3.2 Grid World Experiments

We conducted experiments in the simulated grid world environment, training the four baseline agents and our *Neural-SLAM* agent continuously over a curriculum of 3 courses. We observe that our *Neural-SLAM* agent shows a relatively consistent and stable performance across all 3 courses (Fig. 3). Specifically, our agent can still successfully and reliably explore in the 3rd course where the environments contain more complex structures for which effective long-term

memory is essential (the *A3C-Nav2* agent fails to learn the 1st course in all multiple trainings that we conducted). We visualize the memory addressing in Fig. 4, and observe that the write head addressing weight ω_w converges to a more focused attention to center around the current pose of the agent, while the read head addressing weight ω_r learns to spread out over the entire world area, so that the resulting read vector r can summarize the current memory for the agent to make planning decisions. We note that the memory and the weights for the write/read heads are all initialized to the size of 16×16, since generalization performance tests will be conducted on worlds of those sizes, as will be discussed in Sec. 3.3, yet the agent is able to constrain its writing and reading to the correct map size that it is currently traveling on, which is 8×8 (within the red bounding boxes).

3.3 Generalization Tests

In the experiments discussed above, the agents are trained across 3 courses on world sizes ranging from 8×8 to 12×12. We conducted additional experiments on a set of 50 pre-generated worlds of size 16×16 to test the generalization capabilities of the trained agents. Specifically, we deploy the following agents onto the same set of 50 16×16 worlds, starting from the same position in each world: a *Random* agent which would always select random actions and whose performance can be viewed as a measure of the complexity of the tasks; a trained *A3C-Nav2* agent, as this agent is the most similiar to the model proprosed by (Mirowski et al. 2016) and is shown to be able to generalize its navigation capabilities across different environments; and our trained *Neural-SLAM* agent. We note that no step limit is set for the *Random* agent, while for both the *A3C-Nav2* and *Neural-SLAM* agents, an episode will be terminated if the agent has not finished the exploration task within 750 steps. The experimental results are summarized in Table 1.
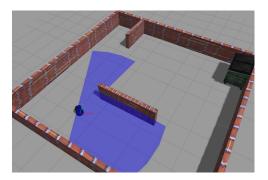
We can observe from Table 1 that the generalization tasks here are relatively challenging as the *Random* agent takes an average of 5531.600 steps to finish an episode. We can also see that the *A3C-Nav2* agent has almost no capability to generalize to much larger environments. We suspect that although the two stacked LSTMs enable the *A3C-Nav2* agent to memorize its odometry, thus it would not travel back to places that it has recently traveled. The lack of an external memory to store its perception of the world makes it difficult to navigate to unexplored areas that are far outside of its current vicinity. While for the *Neural-SLAM* agent, since it embeds a SLAM structure within the planning module, it is able to construct an internal representation of the world, which enables it to identify and plan to go to unexplored areas that might be relatively far away. These different behaviors can be observed in the supplementary video.

3.4 Gazebo Experiments

We also experimented with a simple $3D$ world built in Gazebo. We used a slightly different reward structure: -0.005 as a step cost, -0.05 for collision, 1 for the completion of an exploration, and the exploration reward is scaled down by 0.1 compared to the grid world experiments. We

Table 1: Testing statistics for the generalization experiment, showing the performance of *Random* (random actions), *A3C-Nav2* (very similar to that proposed by (Mirowski et al. 2016)), and *Neural-SLAM* (ours), each evaluated on the same set of 50 randomly generated worlds of size 16x16. The maximum number of steps per episode was 750 steps for both the *A3C-Nav2* agent and our *Neural-SLAM* agent.

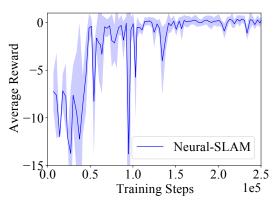|	Steps	Reward	Success Ratio
Random	5531.600 ± 4299.554	-596.644 ± 505.436	-
A3C-Nav2	682.980 ± 201.075	-15.345 ± 11.209	5/50
Neural-SLAM	174.920 ± 174.976	13.732 ± 9.839	46/50

Figure 5: Gazebo experiments (rollout step K: 50; maximum steps per episode: 2500).

deploy 24 learners using docker for training our *Neural-SLAM* agent. From the experimental results shown in Fig. 5, we can see that our *Neural-SLAM* agent is able to solve the task effectively. We also deploy the trained agent in new Gazebo environments to test its generalization performance, and we observe that the agent is still able to accomplish exploration efficiently (we show these experiments in the supplementary video).

4 Conclusions and Future Work

We propose an approach to provide deep reinforcement learning agents with long-term memory capabilities by utilizing external memory access mechanisms. We embed SLAM-like procedures into the soft-attention based addressing to bias the write/read operations towards SLAM-like behaviors. Our method provides the agent with an internal representation of the environment, so as to guide it to make

informative planning decisions to effectively explore new environments. Several interesting extensions could emerge from our work, including the internal reward as discussed in Sec. 2.5, to evaluate our approach on more challenging environments, to conduct real-world experiments, and to experiment with higher dimensional inputs.

References

[Bengio 2013] Bengio, Y. 2013. Deep learning of representations: Looking forward. In *International Conference on Statistical Language and Speech Processing*, 1–37. Springer.

[Bhatti et al. 2016] Bhatti, S.; Desmaison, A.; Miksik, O.; Nardelli, N.; Siddharth, N.; and Torr, P. H. 2016. Playing doom with slam-augmented deep reinforcement learning. *arXiv preprint arXiv:1612.00380*.

[Collett and Graham 2004] Collett, T. S., and Graham, P. 2004. Animal navigation: path integration, visual landmarks and cognitive maps. *Current Biology* 14(12):R475–R477.

[Fischer et al. 2015] Fischer, P.; Dosovitskiy, A.; Ilg, E.; Häusser, P.; Hazırbaş, C.; Golkov, V.; van der Smagt, P.; Cremers, D.; and Brox, T. 2015. Flownet: Learning optical flow with convolutional networks. *arXiv preprint arXiv:1504.06852*.

[Graves et al. 2016] Graves, A.; Wayne, G.; Reynolds, M.; Harley, T.; Danihelka, I.; Grabska-Barwińska, A.; Colmenarejo, S. G.; Grefenstette, E.; Ramalho, T.; Agapiou, J.; et al. 2016. Hybrid computing using a neural network with dynamic external memory. *Nature* 538(7626):471–476.

[Graves, Wayne, and Danihelka 2014] Graves, A.; Wayne, G.; and Danihelka, I. 2014. Neural turing machines. *arXiv preprint arXiv:1410.5401*.

[Gupta et al. 2017] Gupta, S.; Davidson, J.; Levine, S.; Sukthankar, R.; and Malik, J. 2017. Cognitive mapping and planning for visual navigation. *arXiv preprint arXiv:1702.03920*.

[Kingma and Ba 2015] Kingma, D., and Ba, J. 2015. Adam: A method for stochastic optimization. In *Proc. of the International Conference on Learning Representations (ICLR)*.

[Koenig and Howard] Koenig, N., and Howard, A. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 3, 2149–2154. IEEE.

[Latombe 1991] Latombe, J. 1991. *Robot Motion Planning*. Kluwer.

[LaValle 2006] LaValle, S. 2006. *Planning Algorithms*. Cambridge University Press.

[McNaughton et al. 2006] McNaughton, B. L.; Battaglia, F. P.; Jensen, O.; Moser, E. I.; and Moser, M.-B. 2006. Path integration and the neural basis of the 'cognitive map'. *Nature Reviews Neuroscience* 7(8):663–678.

[Mirowski et al. 2016] Mirowski, P.; Pascanu, R.; Viola, F.; Soyer, H.; Ballard, A.; Banino, A.; Denil, M.; Goroshin, R.; Sifre, L.; Kavukcuoglu, K.; et al. 2016. Learning to navigate in complex environments. *arXiv preprint arXiv:1611.03673*.

[Mnih et al. 2016] Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; and Kavukcuoglu, K. 2016. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, 1928–1937.

[Oh et al. 2016] Oh, J.; Chockalingam, V.; Singh, S.; and Lee, H. 2016. Control of memory, active perception, and action in minecraft. *arXiv preprint arXiv:1605.09128*.

[O'keefe and Nadel 1978] O'keefe, J., and Nadel, L. 1978. *The hippocampus as a cognitive map*. Oxford: Clarendon Press.

[Parisotto and Salakhutdinov 2017] Parisotto, E., and Salakhutdinov, R. 2017. Neural map: Structured memory for deep reinforcement learning. *arXiv preprint arXiv:1702.08360*.

[Shen, Michael, and Kumar 2012] Shen, S.; Michael, N.; and Kumar, V. 2012. Autonomous indoor 3d exploration with a micro-aerial vehicle. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, 9–15. IEEE.

[Stachniss 2009] Stachniss, C. 2009. *Robotic mapping and exploration*, volume 55. Springer.

[Sukhbaatar et al. 2015] Sukhbaatar, S.; Weston, J.; Fergus, R.; et al. 2015. End-to-end memory networks. In *Advances in neural information processing systems*, 2440–2448.

[Tai, Paolo, and Liu 2017] Tai, L.; Paolo, G.; and Liu, M. 2017. Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. *arXiv preprint arXiv:1703.00420*.

[Tamar et al. 2016] Tamar, A.; Wu, Y.; Thomas, G.; Levine, S.; and Abbeel, P. 2016. Value iteration networks. In *Advances in Neural Information Processing Systems*, 2154–2162.

[Thrun, Burgard, and Fox 2005] Thrun, S.; Burgard, W.; and Fox, D. 2005. *Probabilistic Robotics*. MIT Press.

[Zhang et al. 2016] Zhang, J.; Springenberg, J. T.; Boedecker, J.; and Burgard, W. 2016. Deep reinforcement learning with successor features for navigation across similar environments. *arXiv preprint arXiv:1612.05533*.

[Zhu et al. 2016] Zhu, Y.; Mottaghi, R.; Kolve, E.; Lim, J. J.; Gupta, A.; Fei-Fei, L.; and Farhadi, A. 2016. Target-driven visual navigation in indoor scenes using deep reinforcement learning. *arXiv preprint arXiv:1609.05143*.