Proceedings of Machine Learning Research vol 144:1–13, 2021

Learning to Actively Reduce Memory Requirements for Robot Control Tasks

Meghan Booker MEBOOKER@PRINCETON.EDU

Anirudha Majumdar ANI.MAJUMDAR@PRINCETON.EDU

Department of Mechanical and Aerospace Engineering, Princeton University

Abstract

Robots equipped with rich sensing modalities (e.g., RGB-D cameras) performing long-horizon tasks motivate the need for policies that are highly *memory-efficient*. State-of-the-art approaches for controlling robots often use memory representations that are excessively rich for the task or rely on hand-crafted tricks for memory efficiency. Instead, this work provides a general approach for *jointly synthesizing* memory representations and policies; the resulting policies *actively seek* to reduce memory requirements. Specifically, we present a reinforcement learning framework that leverages an implementation of the *group LASSO* regularization to synthesize policies that employ low-dimensional and task-centric memory representations. We demonstrate the efficacy of our approach with simulated examples including navigation in discrete and continuous spaces as well as vision-based indoor navigation set in a photo-realistic simulator. The results on these examples indicate that our method is capable of finding policies that rely only on low-dimensional memory representations, improving generalization, and actively reducing memory requirements.

Keywords: Memory-Efficiency, Navigation, Reinforcement Learning

1. Introduction

Consider a robot given a coverage task on a building floor. For example, it could be tasked with performing a safety inspection or collecting data. With the increasing availability and use of high-resolution sensors such as cameras and LiDAR, such tasks require the robot to process high-dimensional observations for real-time decisions. Current navigation approaches typically involve constructing and utilizing a high-fidelity map of the robot's environment (Cadena et al., 2016; Sun et al., 2018; Doherty et al., 2019; Vasilopoulos et al., 2020). However, is a map necessary for the task? Does the map-based representation satisfy the robot's onboard memory constraints? Are there representations that are more memory efficient? These fundamental and practical questions motivate the need to have principled methods for finding memory representations that are not only sufficient for the task at hand but also reduce the robot's memory requirements.

In order to illustrate the potential benefits of memory-efficient policies, consider a robot tasked with covering an $n \times n$ maze (a simplified version of the building floor coverage task). Blum and Kozen (1978) show that there is a control policy — a clever, handcrafted, wall-following and zig-zagging routine — that only utilizes $O(\log n)$ bits of memory. This policy thus requires *significantly less* memory than one that relies on building and using a map of the environment (a map-building strategy requires at least $O(n^2)$ memory). Beyond memory efficiency, such a policy also affords additional important advantages including (i) computational efficiency, and (ii) improved generalization/robustness. For example, Blum and Kozen's policy does not need to perform real-time computations with the entire map as an input. Additionally, a policy that requires $O(\log n)$ memory

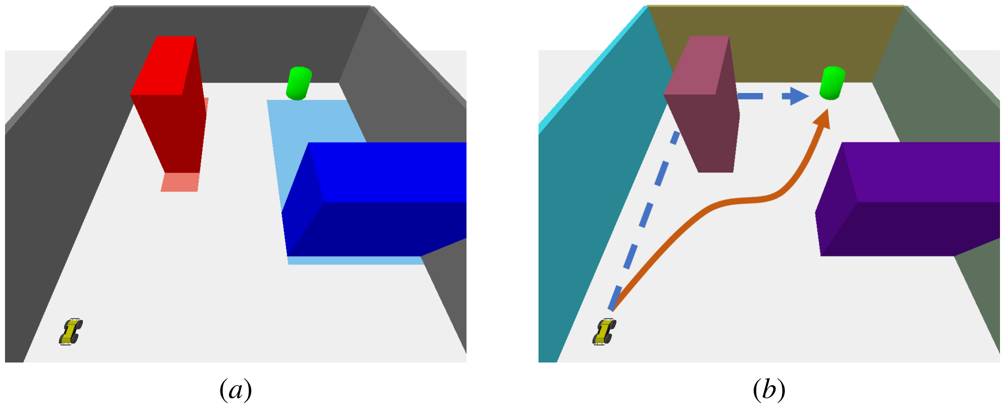

Figure 1: A depiction of the maze navigation problem with continuous state and action spaces described in Section 5.2. The Husky robot needs to navigate from the southwest corner to the goal (green). **(a)** A sample maze used in training. The red and blue obstacles are placed within the respective shaded regions. **(b)** New maze introduced during testing. The paths on the floor illustrate the policies found by a standard policy gradient (PG) method (solid orange) and the approach presented here (dotted blue). The latter (wall-following) policy is significantly more memory efficient and is also robust to changes to the distribution of obstacle colors.

is inherently *task-centric*; irrelevant geometric details of the environment (e.g., the exact positions or colors of obstacles in the environment) do not affect the robot's behavior. The policy can thus be highly robust to uncertainty or noise in these task-irrelevant features.

An important feature of memory-efficient policies is that they can be *qualitatively different* from ones that utilize map-based representations. As a simple example, consider the navigation problem demonstrated in Figure 1. A policy that chooses to follow the wall can be significantly more memory-efficient than one that navigates through the environment diagonally (since the wall-following strategy does not need to maintain information pertaining to obstacle locations). This motivates the need to *jointly synthesize* the memory representation and the control policy; such a joint synthesis can lead to policies that *actively* reduce memory requirements.

Statement of Contributions. The goal of this paper is to synthesize low-dimensional, task-centric memory representations and control policies. Our primary contribution is a reinforcement learning framework for finding policies that achieve *active memory reduction (AMR)*. In particular, we leverage a group LASSO regularization scheme (Yuan and Lin, 2006; Scardapane et al., 2017) to enforce low-dimensional memory representations while simultaneously finding policies via a policy gradient (PG)-style algorithm that we refer to as AMR-PG. To our knowledge, this is the first work to find AMR policies in continuous state and action spaces. Lastly, we demonstrate the efficacy of our approach on three simulated examples that demonstrate our method's ability to find AMR policies that reduce the dimension of the required memory representation and improve generalization as compared to standard PG methods.

2. Related Work

Memory-Efficient Representations. There are several approaches that consider memory-efficient representations for robot navigation tasks including gap navigation trees (Murphy and Newman, 2008; Tovar et al., 2005), compact maps (Srivastava and Michael, 2016), and graph-like topometric maps (Ort et al., 2020). While each of these share this work's goal of memory efficiency, these memory representations are hand-crafted for certain applications or domains. In contrast, we aim to provide a general approach for finding memory-efficient representations and policies. Recent work by O'Kane and Shell (2017) takes a step in this direction by automatically designing minimal memory representations and policies via combinatorial filters. However, their formulation defines memory with respect to the number of nodes in a policy graph and is restricted to discretized state and action spaces. Instead, our work defines memory complexity as the dimension of a continuous representation and is applicable to continuous state and action spaces.

Map-Free Representations in RL. End-to-end reinforcement learning (RL) of policies provides one avenue towards generating task-centric representations that avoid explicit geometric representations such as maps (see, e.g., Levine et al. (2016, 2018); Zhu et al. (2017)). Recurrent neural network (RNN) architectures allow one to incorporate memory into policies learned via RL (Heess et al., 2015). For example, in the context of navigation, Chen et al. (2017) use a long short-term memory (LSTM) (Hochreiter and Schmidhuber, 1997) architecture to navigate mazes with cul-de-sacs. While these approaches are able to find policies that maintain task-centric representations in memory, they do not try to explicitly minimize the memory. In practice, such approaches often choose the dimension of the memory with little to no knowledge of the appropriate size for the task.

Memory-Efficient Representations in RL. Recent work in RL utilizes *self-attention* (Vaswani et al., 2017) before recurrent memory layers. For example, Baker et al. (2019) use this method in their policy architecture to train agents to play hide-and-seek games over a long time horizon. In work by Tang et al. (2020), they highlight the value of self-attention for memory-efficient representations. Specifically, they show how self-attention can be used as a bottleneck to promote the memory representation to only use task-centric features. They also demonstrate that such a bottleneck allows them to only use a small number of memory dimensions, e.g., an LSTM with only 16 memory state dimensions in a third-person perspective navigation task. While this type of approach is capable of finding task-centric representations that are low-dimensional, the memory dimension still needs to be specified *a priori*. In contrast, we present a regularization scheme that explicitly seeks to minimize the memory dimension.

A different line of work learns task-centric memory representations via information bottlenecks (Achille and Soatto, 2018; Pacelli and Majumdar, 2020). These approaches seek policies with "low complexity" as defined in terms of the *information* contained in the memory representation. For example, in work by Pacelli and Majumdar (2020), the objective is to minimize the information content about the state in the memory representation. Our work, instead, defines memory complexity in terms of the dimension; such a measure of complexity is more physically meaningful and tied to the robotic system's onboard memory constraints.

3. Problem Formulation

Our goal is to find a policy that utilizes a low-dimensional, task-centric memory representation. To formalize this, we focus on robot tasks that can be defined with cost functions of the form $\sum_{t=0}^{T} c_t(x_t, u_t)$ where $x_t \in \mathcal{X}, u_t \in \mathcal{U}$, and $y_t \in \mathcal{Y}$ represent the robot's state, control action, and sensor observation at time t respectively. Note that T is fixed *a priori*. The state space $\mathcal{X}$, action space $\mathcal{U}$, and observation space $\mathcal{Y}$ may be continuous or discrete. Additionally, the robot's dynamics and sensor model are described by unknown conditional distributions $p(x_{t+1}|x_t, u_t)$ and $s(y_t|x_t)$ respectively.

We focus on partially-observable settings where the robot may need to choose actions based on the history of observations (i.e., where the Markov property does not necessarily hold if only considering observations). We thus structure our policies in the form $\pi_t(u_t|m_t)$, where m_t is the memory state at time-step t. Here, m_t is a function of the current observation and previous memory state, i.e., $m_t = q_t(y_t, m_{t-1})$. Ideally, the memory state m_t should (i) contain enough information about the sequence $y_1 y_2 \dots y_{t-1}$ of past observations in order choose good actions, and (ii) have minimal dimension d, where $m_t \in \mathbb{R}^d, \forall t = 0, \dots, T$. To formalize the above desiderata, we introduce the matrix zero norm[1].

Definition 1 (Matrix Zero Norm) *Let $a^i \in \mathbb{R}^p$ represent the transposed i-th row of matrix $A \in \mathbb{R}^{n \times p}$. Additionally, let*

$$1(\|a^i\|_0 > 0) := \begin{cases} 0, & \textit{if } \|a^i\|_0 = 0 \\ 1, & \textit{if } 0 < \|a^i\|_0 \leq p \end{cases}$$

indicate if there exists a non-zero element in a^i. Then, the matrix zero norm is defined as the number of non-zero rows, i.e.,

$$\|A\|_0 := \sum_{i=1}^{n} 1(\|a^i\|_0 > 0).$$

Thus $\|M\|_0$, where $M := [m_0 m_1 \dots m_T]$, corresponds to the number of effective dimensions needed by the memory states across the trajectory. If $\|M\|_0 = d < D$, where $m_t \in \mathbb{R}^D$, then the memory representation is effectively reduced from dimension D to d.

To find a memory representation that is both low-dimensional and task-centric, we minimize the memory representation dimension subject to an upper bound on the expected cost of the trajectory:

$$\underset{\substack{q_t(y_t, m_{t-1}) \\ \pi_t(u_t|m_t)}}{\text{minimize}} \quad \big\|M\big\|_0 \quad \text{s.t.} \quad \mathbb{E}\left[\sum_{t=0}^{T} c_t(x_t, u_t)\right] \leq C^{\max}, \tag{1}$$

where $C^{\max} \in \mathbb{R}$ is the maximum allowable expected cost. Since actions are conditioned on the memory state, requiring the matrix zero norm of the memory states to be small means that the policy may need to take actions leading to lower dimensional ones. In other words, the policy actively reduces memory requirements. We call this an *active memory reduction* (AMR) policy.

4. Learning AMR Policies

In this section, we present our approach for the AMR policy synthesis problem (1). We pose the problem as a reinforcement learning problem where q and π are parameterized using neural networks: RNN, $q^w(y_t, m_{t-1})$, connected to a feedforward network that outputs $\pi^w(u_t|m_t)$. We use w to refer to the combined set of weights corresponding to q and π, and the output is treated as a distribution that the control action, u_t, is sampled from; see Figure 2. The primary challenges with (1) then come from (i) the non-differentiability of the matrix zero norm, and (ii) the hard constraint on the expected cost. To tackle these, we relax the matrix zero norm with a regularizer used in group LASSO problems and soften the hard constraint. We discuss these steps, describe our overall policy gradient (PG)-style algorithm, and discuss memory-efficiency below.

1. Note that, like the zero norm for vectors in Euclidean spaces, the matrix zero norm is not a proper norm because it is not homogeneous.

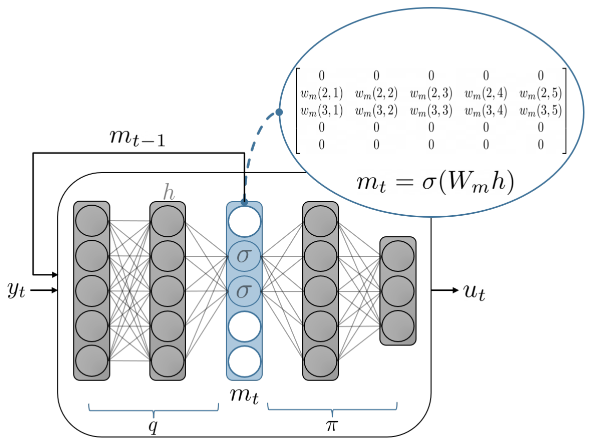
$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ w_m(2,1) & w_m(2,2) & w_m(2,3) & w_m(2,4) & w_m(2,5) \\ w_m(3,1) & w_m(3,2) & w_m(3,3) & w_m(3,4) & w_m(3,5) \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$
$m_t = \sigma(W_m h)$
m_{t-1}
h
y_t
σ
σ
u_t
m_t
q
π

Figure 2: Effective dimensionality reduction of m_t that occurs from applying the AMR regularizer defined in Section 4.1. The regularizer is applied to the incoming weight matrix of the memory layer (blue). The network structure shown takes y_t and m_{t-1} as input and outputs a distribution for u_t to be sampled from.

4.1. Dimensionality Reduction Based on the $l_{2,1}$-norm

A well-known and widely-used convex relaxation of the vector zero norm is the l_1-norm. However, since (1) aims for sparsity of entire matrix rows, this relaxation cannot be used directly. Instead, we use the $l_{2,1}$-norm seen in group LASSO (Yuan and Lin, 2006) to capture this desired behavior. The $l_{2,1}$-norm for matrix $A \in \mathbb{R}^{n \times p}$ is written as:

$$\|A\|_{2,1} := \sum_{i=1}^{n} \|a^i\|_2. \tag{2}$$

Notice that for $A \in \mathbb{R}^{n \times 1}$, (2) is the l_1-norm. Hence, we can expect that minimizing the $l_{2,1}$-norm will promote sparsity of entire matrix rows similar to how minimizing the l_1-norm promotes sparsity of elements in a vector.

The $l_{2,1}$-norm is also effective as a regularizer on groups of weights in neural networks (Scardapane et al., 2017) and for promoting sparsity of hidden states in LSTMs (Wen et al., 2018). Our insight is to now apply it in an RL context to learn memory-efficient policies. First notice, for the time-invariant case, that $\|M\|_0$ is equivalent to the matrix zero norm of the incoming weights at the memory layer as shown in Figure 2. Here, "memory layer" refers to the last layer of a standard RNN, $q^w(y_t, m_{t-1})$, that gives output m_t. More formally, let d_m and d_h be the number of neurons at the memory layer and preceding hidden layer respectively and define the incoming memory layer weight matrix to be $W_m \in \mathbb{R}^{d_m \times d_h}$. We then relax the matrix zero norm with the $l_{2,1}$-norm, i.e., $\|W_m\|_{2,1}$. Intuitively, minimizing this will promote entire rows of W_m to be sparse which in turn, effectively drops out neurons (i.e., dimensions of m_t).

After we relax the matrix zero norm, we soften the hard constraint on the expected cost. Our new reinforcement learning objective then becomes:

$$\underset{w}{\text{minimize}}\ J(w) := \mathbb{E}\left[\sum_{t=0}^{T} c_t(x_t, u_t)\right] + \lambda \Big\|W_m\Big\|_{2,1} \tag{3}$$

where $\lambda \in \mathbb{R}_+$ is a tradeoff parameter between cost and memory efficiency and can be interpreted as the inverse of the Lagrange multiplier.

The regularizer, which we refer to as the AMR regularizer, can additionally be used in time-varying recurrent neural network structures. We achieve this by stacking the memory layer weight matrices to define $\bar{W}_m := [W_{m_0}, W_{m_1}, \dots, W_{m_T}]$ and penalizing $\|\bar{W}_m\|_{2,1}$. This ensures that the same number of memory dimensions are reduced at each time step.

4.2. AMR Policy Gradient Algorithm

Now we describe the algorithm we use to tackle (3). First, we write the gradient of (3) with respect to the network parameters, w, as

$$\nabla_w J(w) = \mathbb{E}\left[\left(\sum_{t=0}^{T} \nabla_w \log \pi^w\Big(u_t | q^w(y_t, m_{t-1})\Big)\right)\left(\sum_{t=0}^{T} c_t(x_t, u_t)\right)\right] + \lambda \nabla_w \Big\|W_m\Big\|_{2,1}. \tag{4}$$

In this form, we extend the canonical policy gradient algorithm, REINFORCE (Williams, 1992), to include the AMR regularizer. We refer to our method as AMR-PG and outline it in Algorithm 1. For network parameter updates, we use the ADAM optimizer (Kingma and Ba, 2014).

Algorithm 1: Active Memory Reduction Policy Gradient (AMR-PG)

```
repeat
    Rollout $N$ trajectories $\{(x_t^n, u_t^n)_{t=0}^T\}_{n=0}^{N-1}$ sampled using $\pi^w\big(u_t|q^w(y_t, m_{t-1})\big)$
    $\nabla_w J(w) \approx \sum_n \big(\sum_t \nabla_w \log \pi^w\big(u_t^n|q^w(y_t^n, m_{t-1}^n)\big)\big)\big(\sum_t c_t(x_t^n, u_t^n)\big) + \lambda \nabla_w \|W_m\|_{2,1}$
    $w \leftarrow w - \alpha \nabla_w J(w)$
until Convergence of J;
```

Once we train a policy using AMR-PG, it remains to determine the reduced memory representation dimension. In our networks used in Section 5, we apply a $\tanh$ nonlinearity to the outputs preceding the memory layer. This allows us to upper bound the value of the memory state at dimension i with the sum of the magnitudes of the incoming weights at dimension i, i.e., $m_t(i) \leq \sum_{j=1}^{d_h} |w(i,j)|$ — we refer to the value of the upper bound as the "memory saliency". Thus as a general rule for determining top contributing dimensions, we discard any dimensions whose memory saliency is at least two orders of magnitude smaller than the highest memory saliency. After determining the dimension reduction, the network can be trained for several epochs with the cut dimensions and regularizer removed. This will provide a hard dimensionality reduction if desired, i.e., explicitly force all incoming weights at a dimension to be zero. In our results discussed in Section 5, we test with the raw trained network to give qualitative insight for how well the memory representation reduced its dependency on task-irrelevant features.

4.3. Discussion on Memory Usage and Reduction

It is important to discuss the efficacy of our approach for reducing a policy's memory requirements. In our approach, we specifically choose to focus on minimizing the RNN memory dimension as this directly determines the complexity of the memory representation for the task (motivated by Blum and Kozen (1978)'s $O(\log n)$ representation versus a map's $O(n^2)$). At initial glance, though, it may seem that the size of the network, i.e., size of RNN q^w and policy π^w, can be arbitrarily large (in terms of number of layers and number of neurons/layer).

This could potentially outweigh memory reductions achieved at the memory layer. However, there are significant memory savings when viewing our approach holistically and applying it in a principled manner. Specifically, consider a simple RNN that has memory size: (dimension of the robot's observations) $\times$ (time horizon). Such a network avoids preemptively losing task-centric information, since the network has the space to copy each observation to memory (i.e., maintain all information the robot has received). As the time horizon increases (a large horizon is realistic for robotic tasks), the size of the memory layer becomes a key contributor to the overall size of the network. Thus, reducing the dimension of this layer becomes impactful for reducing the overall

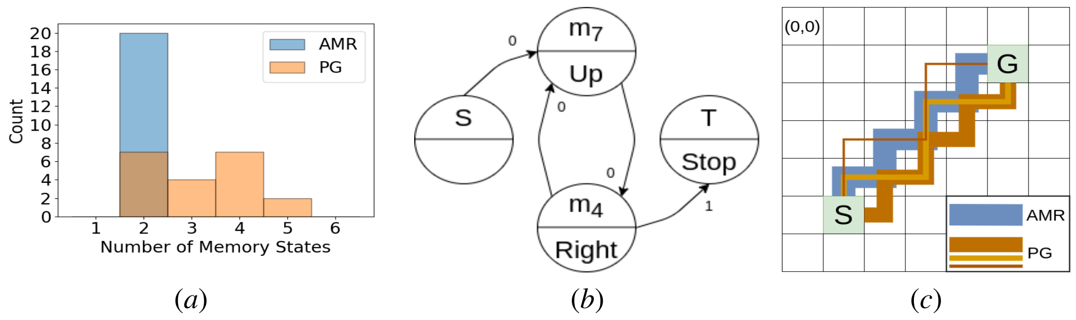

Figure 3: Discrete navigation results. **(a)** Number of memory states needed by AMR-PG and PG policies across 20 seeds. **(b)** A sample memory-optimal Moore machine recovered by AMR-PG where S and T are start and terminal states respectively and m_4 and m_7 are the two memory states used by the AMR-PG policy. The transitions are the robot's observations as described in Section 5.1. **(c)** The paths taken by AMR-PG (blue) and PG (various orange) policies. The line weight indicates the frequency of the path across the 20 seeds.

memory requirements. If the overall network size is a concern, one can apply additional techniques with our method such as regularization (Scardapane et al., 2017), dropout (Srivastava et al., 2014), or distillation (Hinton et al., 2015). However, we emphasize that this is a different objective than ours; we specifically reduce the complexity of the memory representation needed for the task.

While applying our regularizer to the aforementioned network (with dimension: (dimension of the robot's observations) $\times$ (time horizon)) is the principled method for finding a minimal dimension, task-centric memory representation, that network may be impractically large in practice. The network designer may instead choose the memory dimension to be approximately several times (e.g., 2-5x) greater than the observation dimension; this allows for the potential to store several complete observations in memory if needed for the task. This approach is typical for standard RNN design in RL since there is little to no knowledge of what the appropriate memory dimension is.

5. Examples

Here we illustrate the efficacy of our AMR-PG algorithm described in Section 4 with three examples: (i) an illustrative discrete navigation problem, (ii) a continuous navigation problem with synthetic environments, and (iii) vision-based navigation in an apartment using iGibson, a photo-realistic simulator (Xia et al., 2020). In these examples, we show that AMR-PG significantly reduces the dimension of the memory representation, improves generalization, and finds qualitatively different policies (i.e., policies that actively reduce memory) as compared to policy gradient[2] with the same parameterizations. Details regarding the networks and training procedures are discussed for each example in Appendix A of our extended version (EV) (Booker and Majumdar, 2020).

5.1. Discrete Navigation

In this first example, we specialize our method to discrete spaces in order to illustrate policies that achieve AMR. Specifically, we consider an illustrative example from O'Kane and Shell (2017), where a robot must navigate to a goal location in a grid as shown in Figure 3(c). The robot is equipped with a goal indicator, e.g., $y_t = 1$ means that the robot is at the goal. The robot's

2. In our examples, the policy gradient baseline is not intended to have pre-minimized memory. Rather, the maximum RNN memory size, D, is chosen as it is in practice and as described in Section 4.3. To verify the dimensionality reduction achieved by AMR-PG, one potential baseline is to perform a binary search. However, this takes a prohibitively long time as it requires $O(\log D)$ full training runs (whereas AMR-PG accomplishes this in one).

state, x_t, is described by its cell position. Additionally, the robot takes discrete actions $u_t \in \{[-1,0]^T, [0,1]^T, [1,0]^T, [0,-1]^T, [0,0]^T]\}$ corresponding to `up`, `right`, `down`, `left`, and `stop` respectively. The state evolves with dynamics $x_{t+1} = x_t + u_t$. The cost for this scenario is $c_t(x_t, u_t) = \|x_t - g\|_1/\|x_0 - g\|_1$ for $t = 0, \ldots, T$ where the robot is initialized at $x_0 = [5, 1]^T$ and must navigate to goal $g = [1, 5]^T$.

The goal here is to synthesize a policy that takes the form of a deterministic, Moore-style finite state machine as described by O'Kane and Shell (2017) and shown in Figure 3(b). In this context, a memory-optimal policy is defined as one that requires the fewest number of memory states. For this task, an example of a memory-optimal policy is one that simply alternates between actions `up` and `right` until the goal is observed (O'Kane and Shell, 2017). This policy only requires two memory states (not including the starting and terminal states): one for action `up` and one for action `right`; see Figure 3(b). In contrast, a policy that chooses to repeat {`up, up, right, right`} is more complex as it requires keeping track of how many times an action has been applied; such a policy needs at least four memory states. We demonstrate that our approach recovers the memory-optimal two-state policy identified by O'Kane and Shell (2017). However, our method also handles continuous state, action, and observation spaces (considered in subsequent examples).

Training and Results. We model the memory representation with a one-hot encoding vector that indicates which memory state the robot is using (as opposed to the continuous memory states described in Section 4). For the memory representation mapping, $q^w(y_t, m_{t-1})$, we pass the observations to the memory layer of size 10, where 10 is the maximum number of memory states this task could have (a start state, a state for each time step, and a terminal state). We use the β-Softmax activation function on the memory layer with $\beta = 100$ to encourage concentration around one explicit state for m_t. Then we pass m_t to a fully connected layer with 5 neurons activated by a Softmax function. The output is treated as a categorical distribution that we sample the actions from.

We summarize our training results for 20 seeds in Figure 3. To count the memory states used and recover the Moore machine, we took the argmax of the memory and action layer outputs. For each seed, PG found a cost-optimal policy to the goal (see Figure 3(c)) but required between two and five memory states. In contrast, AMR-PG always found the memory-optimal policy.

5.2. Maze Navigation with RGB-Depth Array

Our next example focuses on a differential-drive robot navigating through a maze (Figure 1). The maze is 10m $\times$ 10m with one red and one blue obstacle sampled within the shaded regions indicated in Figure 1(a). The robot is given a fixed linear velocity of 2m/s and has $T = 80$ time steps (0.1s each) to reach the green goal in the upper right corner of the maze. We model the cost as the Euclidean distance between the robot's position and the goal location normalized by the initial distance to the goal for all time steps. Additionally, the robot has control of its angular velocity and is equipped with a 90° fov RGB-depth sensor that outputs colors and depths along 17 rays. The simulations are performed using Pybullet (Coumans and Bai, 2018).

Qualitatively, there are two policies that are sufficient for navigating to the goal: (i) diagonally navigat-

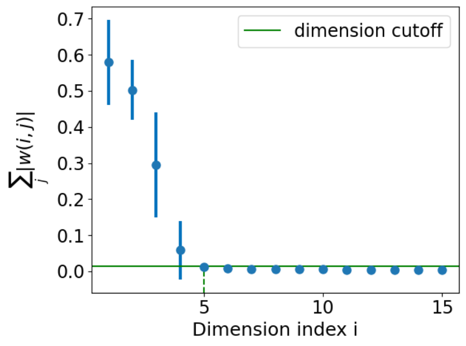

Figure 4: Memory dimension reduction for the maze example. Pictured are the top 15 memory saliency values (see Section 4.2) averaged across five seeds; the error bars represent the standard deviation.

ing through the obstacles to the goal (cost-optimal), and (ii) following the maze wall to the goal (memory-optimal). Notice that the latter policy differs from minimizing the memory needed for the cost-optimal policy. See Figure 1(b) and our video[3] for an illustration of these policies.

Results. We compare AMR-PG with a standard PG method that uses the same neural network parameterization (hyperparameters are provided in the Appendix of the EV). The average cost and final normalized distance to the goal (across five seeds) for training and testing scenarios are summarized in Table 1. For four out of five seeds, PG found the cost-optimal solution of diagonally navigating through the obstacles. (The other seed found the wall-following policy as a result of minimal exploration outside of the far left portion of the maze). In contrast, AMR-PG consistently found the wall-following policy and *significantly reduced* the required memory dimension from 300 to at most 4 as shown in Figure 4. Thus, the policy found by AMR-PG only utilizes at most 1.33% of the memory used by the policy found using PG. We further evaluate the benefits in terms of generalization afforded by our approach. In particular, we test the policies on environments with obstacle colors that differ from ones seen during training. The performance of the PG policies degraded significantly. In contrast, the performance of the policies found using AMR-PG remained almost entirely unaffected. This result combined with the compact memory representation suggests that AMR-PG finds policies for this problem that *actively reduce* memory and only maintain task-centric representations that utilize the distance values to the wall.

Scenario	Policy Gradient		AMR-PG	
	Cost	Dist.	Cost	Dist.
Training	**35.09± 3.22**	**0.11±0.05**	41.95±2.86	0.17±0.11
Testing	**34.99± 3.25**	**0.11±0.04**	42.71±3.36	0.19±0.13
Testing (Swapped Colors)	52.44±5.77	0.53±0.20	**43.93±3.35**	**0.21± 0.14**
Testing (New Colors)	51.03±4.24	0.53±0.12	**43.35±3.38**	**0.21±0.14**

Table 1: Average costs and final normalized distances to the goal across five seeds. We used a fixed training set of 250 mazes and the same 20 testing mazes across three testing instances: (i) same color scheme used in training, (ii) swapped blue and red obstacle colors, and (iii) unseen colors on the walls and obstacles (the specific colors used are shown in Figure 1(b)).

5.3. Vision-Based Navigation

The goal of our last example is to demonstrate AMR-PG's ability to scale to a more realistic scenario: vision-based navigation in a photo-realistic simulation environment. In this example, a TurtleBot is randomly initialized in the hallway of the Placida apartment in iGibson (Xia et al., 2020) and needs to navigate to the kitchen as shown in Figure 5(a). Specifically, the TurtleBot's initial x position is sampled uniformly between the set $[x_0 - 2\text{m}, x_0 + 2\text{m}]$ while the y position and yaw are fixed such that the TurtleBot is centered in the hallway and facing the tables. The cost is described by a weighted sum of a sparse goal reward, a reward for progress towards the goal (as measured by geodesic distance), a collision cost, and an angle (yaw) cost. The control actions specify linear and angular velocities. Additionally, the TurtleBot is equipped with a 90° fov RGB-D camera with a resolution of 128×128. We preprocess these observations with a convolutional neural network before passing them to our AMR network. For more details, see Appendix A in the EV.

Results. For this example, AMR-PG found a significant memory reduction from 100 dimensions down to 34 consistently across five seeds — a memory savings of 66% (see Figure 5(b)).

3. https://youtu.be/x5yYhLoG6jY

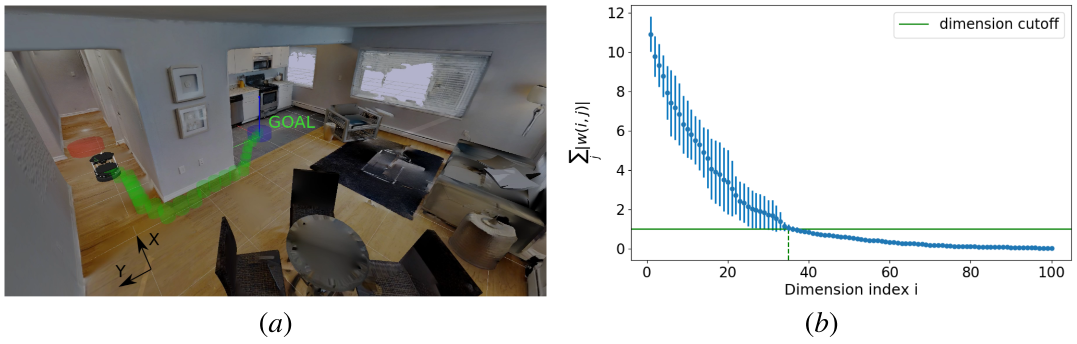

Figure 5: **(a)** Vision-based navigation in Placida apartment in iGibson (Xia et al., 2020). The TurtleBot is randomly initialized in the hallway (red circle) and must navigate to the kitchen (blue circle). The green path visualizes the robot's shortest path to the goal. Both AMR-PG and PG found policies that roughly follow this path. **(b)** Dimensionality reduction achieved by AMR-PG for this task (vision-based navigation). Averaged across five seeds, AMR-PG is able to reduce 100 dimensions down to 34; the error bars represent the standard deviation.

Importantly, these savings did not impact the performance of the policy. On average, AMR-PG obtained a reward of 263.7 ± 34.1 on 20 initial states sampled from the same x range seen in training, while PG obtained a similar reward of 257.6 ± 49.4 on the same initial states. We also initialized the robot from 20 states drawn from an enlarged set of initial conditions: $[x_0 - 2.5\text{m}, x_0 + 2.5\text{m}] \times [y_0 - 0.1\text{m}, y_0 + 0.1\text{m}] \times [\theta_0 - 40°, \theta_0 + 40°]$. In this case, the TurtleBot only collided once using the AMR-PG policies from the five seeds. The policies found using PG resulted in five collisions (using the same set of initial states). Thus, the policies found by AMR-PG achieve effective dimensionality reduction and show potential for improved generalization across different initial conditions. We refer the reader to the video of these results.

6. Conclusion

We presented a reinforcement learning approach for jointly synthesizing a low-dimensional memory representation and a policy for a given task. This joint synthesis allows one to find policies that *actively* seek to reduce memory requirements. The key insight of our approach is to leverage the group LASSO regularization to encourage drop-out of neurons at the memory layer while simultaneously finding policies via a policy gradient approach. We refer to this new algorithm as AMR-PG. Additionally, we demonstrate our approach on discrete and continuous navigation problems, including vision-based navigation in a photorealistic simulator. Comparing AMR-PG and standard PG, we demonstrate that our approach can find low-dimensional representations, improve generalization, and find qualitatively different policies.

Future Work. There are several interesting future directions for this work. One immediate extension is to find AMR policies with actor-critic architectures (e.g., using PPO (Schulman et al., 2017)), and more complex memory network architectures (e.g., LSTMs (Hochreiter and Schmidhuber, 1997)). On the practical front, we are excited to work towards the employment of AMR policies on resource-constrained robotic platforms such as micro aerial vehicles. An important step for this is demonstrating that the AMR policies scale well to long-horizon tasks. Lastly, a particularly exciting direction is to explore whether our approach leads to policies that are more *interpretable* (since they only maintain low-dimensional memory representations) by visualizing features that impact the memory representation (e.g., using saliency maps (Simonyan et al., 2013)).

Acknowledgments

This work is partially supported by the National Science Foundation [IIS-1755038], and the School of Engineering and Applied Science at Princeton University through the generosity of William Addy '82.

References

Alessandro Achille and Stefano Soatto. A separation principle for control in the age of deep learning. *Annual Review of Control, Robotics, and Autonomous Systems*, 1:287–307, 2018.

Bowen Baker, Ingmar Kanitscheider, Todor Markov, Yi Wu, Glenn Powell, Bob McGrew, and Igor Mordatch. Emergent tool use from multi-agent autocurricula. *arXiv preprint arXiv:1909.07528*, 2019.

Manuel Blum and Dexter Kozen. On the power of the compass (or, why mazes are easier to search than graphs). *19th Annual Symposium on Foundations of Computer Science*, 1978.

Meghan Booker and Anirudha Majumdar. Learning to actively reduce memory requirements for robot control tasks. *arXiv preprint arXiv:2008.07451*, 2020.

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

Steven Chen, Nikolay Atanosov, Arbaaz Khan, Konstantinos Karydis, Daniel Lee, and Vijay Kumar. Neural network memory architectures for autonomous robot navigation. *arXiv preprint arXiv:1705.08049*, 2017.

Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning, 2018.

Kevin Doherty, Dehann Fourie, and John Leonard. Multimodal semantic SLAM with probabilistic data association. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pages 2419–2425, 2019.

Nicolas Heess, Jonathan J. Hunt, Timothy P. Lillicrap, and David Silver. Memory-based control with recurrent neural networks. *arXiv preprint arXiv:1512.04455*, 2015.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8): 1735–1780, November 1997.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research (IJRR)*, 37(4-5):421–436, 2018.

Liz Murphy and Paul Newman. Using incomplete online metric maps for topological exploration with the gap navigation tree. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2008.

Jason M. O'Kane and Dylan A. Shell. Concise planning and filtering: Hardness and algorithms. *IEEE Transactions on Automation Science and Engineering*, 14(4):1666–1681, 2017.

Teddy Ort, Krishna Murthy, Rohan Banerjee, Sai Krishna Gottipati, Dhaivat Bhatt, Igor Gilitschenski, Liam Paull, and Daniela Rus. Maplite: Autonomous intersection navigation without a detailed prior map. *IEEE Robotics and Automation Letters*, 5(2):556–563, 2020.

Vincent Pacelli and Anirudha Majumdar. Learning task-driven estimation and control via information bottlenecks. In *Proceedings of Robotics: Science and Systems (RSS)*, 2020.

Simone Scardapane, Danilo Comminiello, Amir Hussain, and Aurelio Uncini. Group sparse regularization for deep neural networks. *Neurocomputation*, 241(C):81–89, 2017.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Shobhit Srivastava and Nathan Michael. Approximate continuous belief distributions for precise autonomous inspection. In *Proceedings of the IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pages 74–80, 2016.

Ke Sun, Kelsey Saulnier, Nikolay Atanasov, George Pappas, and Vijay Kumar. Dense 3-D mapping with spatial correlation via gaussian filtering. In *Proceedings of the American Control Conference (ACC)*, pages 4267–4274, 2018.

Yujin Tang, Duong Nguyen, and David Ha. Neuroevolution of self-interpretable agents. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. ACM, 2020.

Benjamin Tovar, Luis Guilamo, and Steven M. Lavalle. Gap navigation trees: Minimal representation for visibility-based tasks. *Algorithmic Foundations of Robotics VI, Springer Tracts in Advanced Robotics*, 17:425–440, 2005.

Vasileios Vasilopoulos, Georgios Pavlakos, Sean L. Bowman, J. Diego Caporale, Kostas Daniilidis, George J. Pappas, and Daniel E. Koditschek. Reactive semantic planning in unexplored semantic environments using deep perceptual feedback. *IEEE Robotics and Automation Letters*, 5(3): 4455–4462, 2020.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the International Conference on Neural Information Processing Systems*, pages 6000–6010, 2017.

Wei Wen, Yuxiong He, Samyam Rajbhandari, Minjia Zhang, Wenhan Wang, Fang Liu, Bin Hu, Yiran Chen, and Hai Li. Learning intrinsic sparse structures within long short-term memory. In *Proceedings of the International Conference on Learning Representations*, 2018.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3–4):229–256, 1992.

Fei Xia, William B. Shen, Chengshu Li, Priya Kasimbeg, Micael Edmond Tchapmi, Alexander Toshev, Roberto Martín-Martín, and Silvio Savarese. Interactive Gibson benchmark: A benchmark for interactive navigation in cluttered environments. *IEEE Robotics and Automation Letters*, 5 (2):713–720, 2020.

Ming Yuan and Yi Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society Series B*, 68:49–67, 2006.

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 *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3357–3364. IEEE, 2017.