Learning a Semantic Prior for Guided Navigation

Yi Wu[1], Yuxin Wu[2], Georgia Gkioxari[2], Yuandong Tian[2], Aviv Tamar[1], Stuart Russell[1]

[1]UC Berkeley [2]Facebook AI Research

Abstract. Learning generalizable agents that can adapt to unseen environments remains an open problem in reinforcement learning. We consider visual navigation and address this problem by utilizing latent semantic regularity in human-designed 3D environments, aiming for generalization across scenarios that are visually diverse but semantically consistent. During training, the agent learns subpolicies to reach different semantic concepts, such as 'move towards the kitchen', and a prior distribution over their pairwise relationships, such as 'kitchen is close to dining room', in the form of a probabilistic graphical model. When testing on new scenarios, the agent dynamically updates its belief of the underlying semantic relationships during exploration and plans its route accordingly towards the final target in an interpretable manner. Our guided navigation method outperforms strong baselines which do not explicitly plan using the semantic content.

Keywords: Reinforcement Learning, Generalization, Navigation, Graphical Model, Semantic Relationships

1 Introduction

Although deep reinforcement learning (DRL) has witnessed several achievements in recent years [21,16,11], the generalization capability of DRL agents is still an open problem. In their majority, RL agents are tested on the same training environments. In contrast, humans show strong adaptation to unknown environments (typically after a few trials and explorations).

Finding a well-defined metric that captures the generalization/adaptation capability of RL agents is not easy. Naively separating environments into a train and a test set is not optimal, due to their heterogeneous properties (e.g., different games in Atari have dramatically different or even irrelevant goals). On the other hand, environments that only slightly differ in appearance (e.g. family of environments with color/shape/texture perturbations of objects [24]) are usually quite narrowly scoped and do not cover diverse real-world scenarios. Ideally, we want a family of environments that appear to be diverse but still share *semantic* properties - the types of objects/rooms and the dynamics between them are the same but their attributes (apperance, color, shape etc.) and the scene layout

drastically differ across scenes. Such environments allow us to test whether an agent can learn such properties and generalize well.

To this end, navigation tasks within human-designed 3D environments provide an effective testbed to measure semantic level generalization. Agents learn from a variety of observations due to different scene layouts, object types, sizes and connectivity, which are ultimately tied to the same underlying natural properties. When agents are faced with the task of achieving semantic goals (e.g., *go to kitchen*), their ability to adapt to different scenes is tested. House3D [26] is an example of such an environment. House3D leverages the SUNCG dataset [22] and contains 45K *human-designed* real-world 3D house models, ranging from single studios to houses with gardens, in which objects are fully labeled with common room and object categories (e.g., bedroom, sofa, etc). Agents trained in these environments see the same set of object categories and underlying targets (or tasks to accomplish).

Different from traditional deep RL agents which learn control *solely* from the high-dimensional continuous visual signal, in this work, we assume that some low-dimensional discrete semantic signals from perception is also available (e.g., the output of a object detector or scene classifier). With this assumption, we propose a *guided navigation* approach: besides training target driven subpolicies from visual signals, based on the semantic signals, we learn a Bayesian model over the semantic structures to provide exploration guidance for the subpolicies. The learned Bayesian model (1) represents a structured prior over the semantic properties of the environments, (2) allows efficient inference and planning when exploring new environments, and (3) is easy to learn and fully interpretable.

Our extensive experiments show that our proposed approach performs strongly: it is better than model-free holistic policies and even slightly outperforms RNN-based meta-controllers, which are trained with the same subpolicies and input signals as our method but have orders of magnitude more parameters and are less interpretable.

2 Related Work

The problem of navigation has been studied extensively [10]. Classical approaches build a 3D map of the scene using SLAM, which is subsequently used for planning [4]. More recently, end-to-end approaches have been applied to tackle various domains, such as maze [13], indoor scenes [27] and Google street view [12].

Evidently, navigation performance deteriorates as the agent's distance from the target increases [27,26]. To aid navigation and boost performance, auxiliary tasks[13,7] are often introduced during training. Another direction for visual navigation is to use a recurrent neural network and represent the memory in the form of a 2D spatial map [9,18,23,5] such that a differentiable planning computation [9,23,5] can be performed on the spatial memory to produce better performances. Our approach considers more general graph structures beyond 2D grids and captures pairwise relationships between *semantic* entities, which

we use as an informative latent structure in semantically rich environments like House3D.

Similar to our work, Savinov *et al.* [20] constructs a graph of nodes corresponding to different locations of the environment. However, they rely on a pre-exploration step within the test scene. In our work, we learn a semantic prior based on the observation that semantic concepts and basic structural rules are shared across real-world scenes. This allows us to directly start solving for the task at hand without any exploratory steps.

Besides task complexity, deep RL agents can fail to adapt to new unseen scenarios. Most deep RL agents are tested in the same training environments [13], disregarding generalization. While limited, different approaches have been proposed to enforce an agent's generalization ability. In [24], domain randomization is used to increase an agent's robustness. In [17,18], randomly generated mazes are used to check the generalization capacity of the trained agent. In our work, we address this issue directly by using a separate test set of environments of different scene layouts such that an agent can not resort to memorization or simple pattern matching to solve the task.

Another direction in RL tackles the problem of fast adaptation to novel environments via meta-learning [3,2,14]. Methods include learning a good parameter initialization for gradient descent [3] or learning a neural network that can fast adapt its policy after exploration in a new environment [2,14]. In contrast, we propose to learn a Bayesian model over the semantic structure of an environment and update the posterior structure via Bayes rule. Our approach (1) can work even without any exploration steps in a new environment and (2) is interpretable and can be potentially combined with any graph-based planning algorithm from the classical AI literature.

Our work can be viewed as a special case of hierarchical reinforcement learning (HRL). Unlike other approaches [25,1], in our work high-level planning is done based on the semantic signal. With orders of magnitudes fewer parameters, our approach is easier to learn compared to recurrent controllers.

Our approach assumes a discrete semantic signal in addition to the continuous state. Similar assumption is also adopted in [19], where the discrete signals are used to tackle the sparse reward problem. The schema network [8] further assumes that even the high dimensional visual signal can be completely represented in a binary form and therefore directly runs logical reasoning on the binary states for accurate forward simulation.

3 Background

We consider the task of multi-target navigation. The agent is given one of K different targets (concepts), $\mathcal{T} = \{T_1, T_2, \ldots, T_K\}$, and is asked to find an instance of that target concept in a 3D environment (e.g., predefined room types, such as kitchen or bedroom, in RoomNav task defined in [26]).

Environment: We define *context-dependent* environment $E(c)$ as a *contextual Markov Decision Process* [6] $E(c) = (\mathcal{S}, \mathcal{A}, P(s'|s, a; c), r(s, a; T, c))$, where

$\mathcal{S}$ is the state space, $\mathcal{A}$ the action space, T the given target. The context c denotes the objects, layouts and any other *semantic* information describing the environment. c is sampled from $\mathcal{C}$, the distribution of possible house arrangements, or the House3D environments. In the experiments, we sample a disjoint partition of a training set $\mathcal{E}_{\text{train}} = \{E(c_i)\}_i$ and a testing set $\mathcal{E}_{\text{test}} = \{E(c_j)\}_j$, where $\{c_i\}$ and $\{c_j\}$ are samples from $\mathcal{C}$.

State: The agent receives a state $s = [s_o, s_s]$. The *observation* s_o, is the image perceived by the first-person view of the agent [1]. The semantic signal s_s is a binary vector that encodes semantic information of objects and layouts.

Transition: The transition probability $P(s'|s, a; c)$ describes the change of a state s given an action a. The context c encodes the position and layouts of the objects in the house, and thus affects the transition. The agent's *physical* dynamics is not changed for different c.

Reward Function: $r(s, a; c, T_i)$ is the reward function for task T_i in $E(c)$. To facilitate training, we design a shaped reward function for the agent r_{train} using the semantic information of scenarios. During testing, the agent is not given any reward signal and succeeds when reaching the target.

Objective: Let $\mu(a|s; \theta)$ denote the agent's deterministic policy parameterized by θ given the state s. The objective is to maximize the testing performance, i.e., the accumulative reward r_{test},

$$\theta^\star = \arg\max_\theta \mathbb{E}_{T_i \sim \mathcal{T}, c \sim \mathcal{C}} \left[R_{\text{test}}(\mu(\theta); c, T_i) \right]. \tag{1}$$

The learned policy is measured with Eq. 1 in the test environments $\mathcal{E}_{\text{test}}$, which is not accessible during training.

4 Learning a Semantic Prior for Guided Navigation

The key idea in our approach is the fact that while each environment can be different in visual appearance and layouts, there are structural similarities between environments that can be captured as a probabilistic graphical model over the semantic information. For example, in most houses the dining room connects to the kitchen, and living room may be next to the entrance. Thus, while navigating in a new, unseen, house, we can use Bayesian inference to estimate the structure of our current house, using the data seen during training as a prior, and use the inferred house plan to efficiently explore the house. For example, if our task is to find the kitchen while the agent is in the living room, we know that the dining room could be a way point along our trajectory. See Sec. 4.1 for details.

We aim to learn a probabilistic model $M^\star(\mathcal{D}, c)$ that captures the distribution $\mathcal{C}$ of contexts, which encodes the semantic information of the environments, from the agent's experiences $\mathcal{D}$ (*i.e.*, the exploration trajectories). Given a new environment $E(c')$, the agent computes a posterior $P(c'|\mathcal{D}', M^\star)$ of its unknown current semantic context c' driven by the learned model $M^\star$ and its current

[1] In this work we consider image input, but other sensory observations can be handled similarly.

observations $\mathcal{D}'$ in $E(c')$. This allows the agent to plan according to its belief of c' in order to reach the goal more effectively. Note that the Bayesian model allows the agent to perform probabilistic inference even with *limited* (or even no) exploration experiences.

Learning an accurate and complete Bayesian model $M^\star(\mathcal{D}, c)$ can be challenging. For 3D navigation, learning a full $M^\star$ would involve 3D reconstruction of the unknown environment from the semantic relationships the agent has observed. In fact, lower level attributes of the scene, such as the color or shape of the rooms and objects, are less relevant for navigation. Therefore, we build a semantic model which disregards the irrelevant information in $\mathcal{D}$. We learn an *approximate* latent variable model $M(y, z; \psi)$ parameterized by ψ with observation variable y and latent variable z. With this simplified model, we aim to extract samples of y only via the semantic signal s_s without using image observation s_o, and ensure z contains semantic information from the underlying context c that are necessary for the tasks in question.

4.1 Definition of a Graphical Model over Target Concepts

Navigating to faraway targets in real 3D scenes is challenging: an agent can get stuck or diverge from the target due to lack of visual evidence about its relative location to the goal. We use the structure of natural scenes to decompose the task into a set of subtasks, each of which requires navigating to a nearby target on the path to the actual goal. In RoomNav, we define these subtasks as finding the neighboring room on the way to the final target. If the connectivity graph of the scene is known a priori, the agent can easily decompose the task into a sequence of distinct subtasks, each achieved with a small number of steps. However, the connectivity graph is not known to the agent beforehand. We wish to build a model that is used by the agent to efficiently navigate to the target.

To this end, we represent our model M as a probabilistic *graph* over the K targets $\{T_1, \ldots, T_K\}$. In RoomNav, each target corresponds to a room type. We use a Bernoulli random variable $z_{i,j}$ to denote whether room type T_i and T_j are connected, *i.e.*, they are in proximity of one another [2]. The random variables $z_{i,j}$ are symmetric, *i.e.*, $z_{i,j} = z_{j,i}$. For simplicity, we assume the edges in the graph are *independent*: we have $z_{i,j} \sim \text{Bernoulli}(\psi^z_{i,j})$, where ψ^z correspond to the model's parameters associated with connectivity. During exploration and for each latent variable $z_{i,j}$, the agent receives samples from a noisy observation $y_{i,j}$ defined as $y_{i,j} \sim z_{i,j} + \epsilon(\psi^y_{i,j})$, where ψ^y are the model's parameters associated with the observations. More concretely,

$$y_{i,j} \sim \begin{cases} \text{Bernoulli}(\psi^y_{i,j,0}) & \text{if } z_{i,j} = 0 \\ \text{Bernoulli}(1 - \psi^y_{i,j,1}) & \text{if } z_{i,j} = 1 \end{cases} \tag{2}$$

Eq. 2 states that the observation $y_{i,j}$ may have a different value from true value $z_{i,j} = c$ with probability $\psi^y_{i,j,c}$. This models the practical case where the agent

[2] Although there can be multiple instances for each type (e.g., a house with two bedrooms), for simplicity we only consider connectivity over types here and leave a more precise graphical model for future work.

may collect wrong connectivity signals due to its imperfect policy (*e.g.*, even for nearby rooms the agent can get stuck and never reach T_j from T_i) or noise in s_s.

We wish to learn the optimal set of parameters $\psi^\star$ of M such that our model $M(y, z; \psi^\star)$ leads to accurate navigation. More specifically, we aim to learn ψ such that (1) the prior distribution $P(z; \psi^\star)$ captures the *semantic* knowledge of the context distribution $\mathcal{C}$ (e.g., garage is outdoor; dining room often connects to kitchen); (2) computing the posterior $P(z|\mathcal{Y}; \psi^\star)$ with Bayes rule reflects the true underlying semantic structure of the unseen environment $E(c')$ after obtaining a set $\mathcal{Y}$ of samples to random variable y.

There are several advantages of representing M in the form of a graph with independent random edges: (1) We can combine any graph-based planning algorithms from the classical AI literature to potentially handle much more complicated tasks beyond visual navigation (e.g., reasoning); (2) the graph representation is naturally interpretable by humans during planning. In the experimental section, we will show that how our graph representations leads to improved navigation performance.

4.2 Combining the Graphical Model with Subpolicies for Guided Navigation

Here we focus on how to obtain samples to random variable y from the semantic signal s_s. Such a low-dimensional discrete signal is commonly used in AI, *e.g.*, for robotic manipulation tasks s_s can indicate whether the robot is holding an object, for video games it is the game status of a player, in visual navigation it can indicate whether the agent reached a landmark. In our setup, we define s_s as a K-bit vector, whose i-th bit indicates whether the agent is now at target T_i.

For simplicity, we assume that s_s is directly given by the environment. Alternatively, one can train an classifier to predict the room type from the agent's observations. In practice, even when the semantic signal is directly given by the environment, s_s can be noisy due to the labeling error[3].

Suppose the agent explores the current environment for a *short* horizon of N steps obtaining N different visual signals ${s_o}^{(1)}, \ldots, {s_o}^{(N)}$ as well as the binary semantic signals ${s_s}^{(1)}, \ldots, {s_s}^{(N)}$. We compute the bit-OR operation over these binary vectors $B = {s_s}^{(1)}$ `OR` $\ldots$ `OR` ${s_s}^{(N)}$. Then for all pair of targets (T_i, T_j) with $B(i) = B(j) = 1$, we assume that T_i and T_j are nearby, which in turn implies a *positive* sample of $y_{i,j}$, For all (T_i, T_j) with $B(i) \neq B(j)$, we assume that the targets are faraway, which implies a *negative* sample of $y_{i,j}$. With these samples of $y_{i,j}$, we can compute the posterior distribution of the latent structure z using the Bayes rule. The posterior is then used to plan on the environment by executing the subtask whose target is towards the final target T_i with the

[3] In the SUNCG dataset on which our environment is built, labels for room regions can be noisy. For example, the corresponding region for a kitchen can be much larger than the actual area. This means that $s_s(i) = 1$ may imply that the agent is *near* room T_i.

highest probability. Concretely, suppose the current state is $s^{(t)}$, we find a most likely trajectory $p_0, \ldots, p_m$ towards the target by Eq. 3 and then set T_{p_1} as our next subtask:

$$\{p_j\}_j = \arg\max_{p_0,\ldots,p_m} P\left[{s_s}^{(t)}(p_0) \left(\prod_{u=1}^{m} z_{p_{u-1},p_u} \right) z_{p_m,i} = 1 \,|\, \mathcal{Y}, M(\psi) \right]. \quad (3)$$

4.3 Learning the Graphical Model

The model's parameters ψ can be decomposed into two parts, ψ_z determines the *prior* of the connectivity between targets; ψ_y affects the observation noise and is related to the performance of the trained neural subpolicy $\mu(\theta)$: if $\mu(\theta)$ has very high success rate for reaching nearby targets, ψ_y should be low; when $\mu(\theta)$ is poor, ψ_y can be relatively higher. Hence, we propose two objectives to learn these two parts of parameters.

Learning ψ_z: Here we consider learning ψ_z from $\mathcal{E}_{\text{train}}$. During training, we have full access to the labels of the environments, and thus know directly the connectivity between any two rooms (T_i, T_j) (e.g., distance within some threshold) for a particular house $E(c) \in \mathcal{E}_{\text{train}}$: if T_i and T_j are connected, we obtain a *positive* sample of $z_{i,j}$; otherwise we obtain a *negative* samples of $z_{i,j}$.

Suppose $\mathcal{Z}$ denotes the samples we obtained for latent variable z from all the training environments. We run maximum likelihood estimate to learn the optimal ψ_z by maximizing

$$L_{\text{MLE}}(\psi_z) = P(\mathcal{Z}|\psi_z). \quad (4)$$

An alternative approach for collecting $\mathcal{Z}$ is to perform random exploration from room T_i to T_j for a long horizon to obtain a sample of $z_{i,j}$[4]. In practice, random exploration is more efficient and can lead to slightly better performances.

Learning ψ_y: Note that there is no supervision for the parameters ψ_y. We can only evaluate the effectiveness of a particular ψ_y by running the guided navigation method in the previous section. Suppose $R_{\text{test}}(\mu(\theta), M(\psi); c, T)$ denotes the accumulative reward on $E(c)$ for target T by applying the guided navigation approach. In a another disjoint validation set of environments $\mathcal{E}_{\text{val}}$, we optimize the accumulative reward as follows

$$L_{\text{valid}}(\phi_y) = \mathbb{E}_{E(c)\in\mathcal{E}_{\text{valid}}, T\sim\mathcal{T}} \left[R_{\text{test}}(\mu(\theta), M(\psi); T, c) \right]. \quad (5)$$

Explicitly optimizing L_{val} is hard. Instead, we apply local search to find the optimal parameters ψ_y.

[4] We can also use the trained policy $\mu(\theta)$ to collect samples. Random exploration is just simpler and we can then separate graph learning and subpolicy training. It will take exponential number of steps for a random policy to reach a faraway target so it is a natural choice to use random exploration to measure "connectivity" between two targets

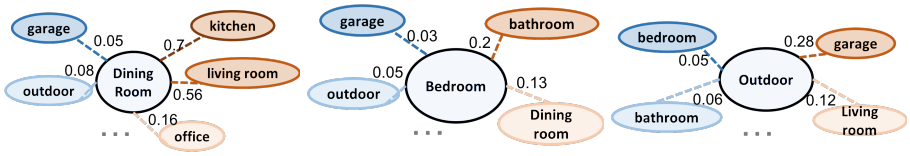

Fig. 1. The most and least likely nearby rooms for dining room (L), bedroom (M) and outdoor (R). Numbers are prior probability of connectivity between two room types.

4.4 Learning the Neural Subpolicy

For the neural subpolicy $\mu(\theta)$, we train K different LSTM policies, i.e., $\mu(\theta_i)$ for each target $T_i \in \mathcal{T}$, using A3C [15]. When navigating towards target T_i, we simply execute the corresponding policy $\mu(\theta_i)$. We notice that training separate sub-policies leads to better performance than training a multi-target policy as originally described in [26].

5 Experiments

In this section, we experiment on the RoomNav task [26] and try to answer the following questions: (1) What does the prior distribution look like? Is it reasonable? (2) Does our graph based guided navigation improve test performances? (3) How does our approximate Bayesian sampling compare with a 'blind' model for parsing the semantic signal using a RNN? (4) Can we extend our approach to more target concepts, i.e., objects? The answers follow.

5.1 Environment and Task Setup

We select $K = 8$ room types as the targets. For training each subpolicy $\mu(\theta)$, we use the same success measure, reward shaping and policy architectures. We use A3C with curriculum learning. Note that $\mu(\theta)$ only takes in the visual observation s_o. For training environments, we use the large set of 200 houses from Hosue3D as $\mathcal{E}_{\text{train}}$ and the test set of 50 houses as $\mathcal{E}_{\text{test}}$. We use the small set of 20 houses as $\mathcal{E}_{\text{val}}$ to cross validate ψ_y. More training details can be found in the supplementary.

In the following experiments, we focus on a fixed horizon H: if the agent does not reach the target within H steps, we terminate the episode and declare it a failure. During evaluation, we run a total of 5000 testing episodes. We also fix the random seed of the environment for a fair comparison so that in each testing episode, the agent will always start from the same location and have the same target.

5.2 The Learned Graph

We visualize the learned parameter ψ_z which defines the prior probability $P(z|M(\psi_z))$. Fig. 1 shows 3 room types and their most and least likely connected rooms

according to the prior probabilities. The learned prior indeed captures reasonable relationships over room types: bathroom is likely to connect to a bedroom; kitchen is often near a dining room while garage is typically outdoor.

5.3 Performance on RoomNav

We compare the performance of our guided navigation approach against two baselines (1) the random policy (denoted by "random") and (2) only executing a single subpolicy $\mu(\theta_i)$ for the target T_i throughout the episode (denoted by "pure $\mu(\theta)$").

We experiment with different (1) exploration steps N and (2) horizon length H. We show performance of different approaches with $N = 30, 50$ and $H = 300, 500, 1000$ in Fig. 2 where the x-axis is the distance in meters[5] of the agent's birth place to the target room while the y-axis is the success rate.

We observe that random policy performs poorly while our guided navigation approach works much better than the pure neural policy, especially on faraway targets: when the horizon becomes longer, the success rate for targets that are further away significantly increases while the single neural policy fails to reach faraway targets even with a large number of steps. This implies that our graphical model based planner provides effective intermediate sub-goals which help the neural policy eventually reach the distant target. Moreover, we notice that for a fixed horizon H with small exploration step, ($N = 30$), our approach leads to better performance. Note that smaller N implies more planning computations and sub-goals. This reflects the importance of having a planning module in navigation.

Fig. 3 shows an example of a success trajectory with our graph-based planner. We visualize the progression of the episode, describe the plans and show the updated graph after exploration.

5.4 Comparing to RNN Baselines without a Graph Representation

To show the effectiveness of our graph-based planner, we train two baseline models by augmenting the pure policy with the semantic signal and by replacing the graph with a memory-based controller.

Augmented policy $\mu_s(\theta_s)$: We train a neural policy $\mu_s(\theta_s)$ which takes as input the semantic signal s_s in addition to the visual signal s_o. Note that these are the same input signals used by our approach when learning the graph.

RNN controller: The graphical model M (Eq. 3) only depends on (1) the semantic signal of the current state ${s_s}^{(t)}$, (2) the final target T_i and (3) the accumulative bit-`OR` feature B from the last exploration period. Hence, we replace M with an LSTM controller that outputs the next subtask and takes in *exactly the same* input signals as M. We also feed in the the previous subtask

[5] Moving towards the target for 1 meter typically requires 2 to 4 actions in the shortest path. The agent also needs to keep staying in the room for additional several steps in order to succeed in the RoomNav task.

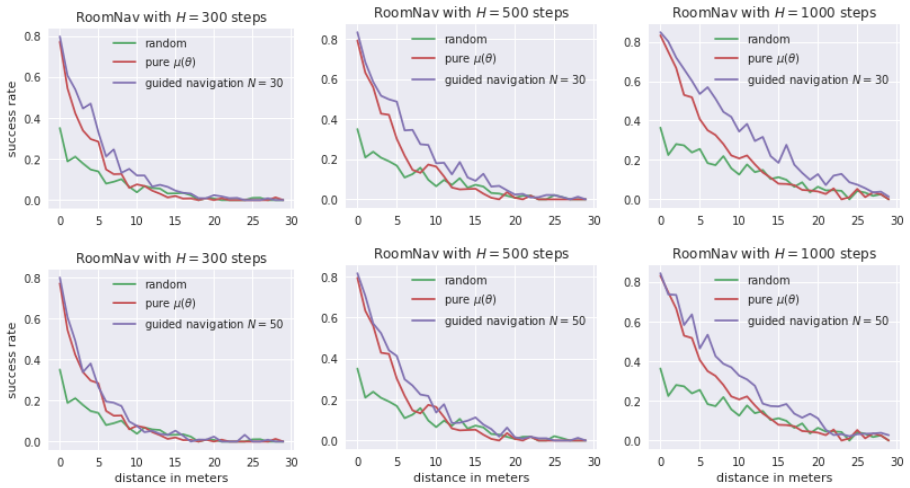

Fig. 2. Performance on RoomNav of random policy (green), pure $\mu(\theta)$ (red) and our guided navigation approach (purple) with different exploration steps N (top: $N = 30$; bottom: $N = 50$). X-axis: meters from agent's birth place to target; Y-axis: success rate.

into the LSTM. We train the LSTM controller using A3C with the *same* neural policy $\mu(\theta)$ used in our approach.

Since both of these baselines use the same information as our graph-based approach, and are trained using RL to maximize performance, they provide fair comparisons to our method. More importantly, the RNN controller has exactly the same input and output space as our graph-based planner and uses two order of magnitude more parameters than our approach. Thus we expect it to perform competitively to our graph-based approach.

For the RNN controller, we train a LSTM with 50 hidden units. For $\mu_s(\theta_s)$, we use the same architecture as the original policy $\mu(\theta)$. We fix $N = 30$ and experiment with different horizons H. The results are shown in Fig. 4. $\mu_s(\theta_s)$ does not improve the original neural policy $\mu(\theta)$. Our approach is comparable with the RNN controller for short horizons and slightly outperforms the RNN controler baseline under long horizon. When comparing to the RNN controller, our approach has the following advantages: (1) M can be learned more efficiently and has much fewer parameters: an LSTM with 50 hidden units has more than 10^4 parameters while M only has 38 parameters[6]; (2) M can easily adapt to different subpolicies $\mu(\theta')$ with little finetuning (ψ^z remains unchanged) while the RNN controller needs to re-train; (3) the model M as well as the planning procedure are fully interpretable.

[6] In practice, for $c \in \{0, 1\}$, we can assign the same value to all $\psi^y_{i,j,c}$. See more in supplementary materials.

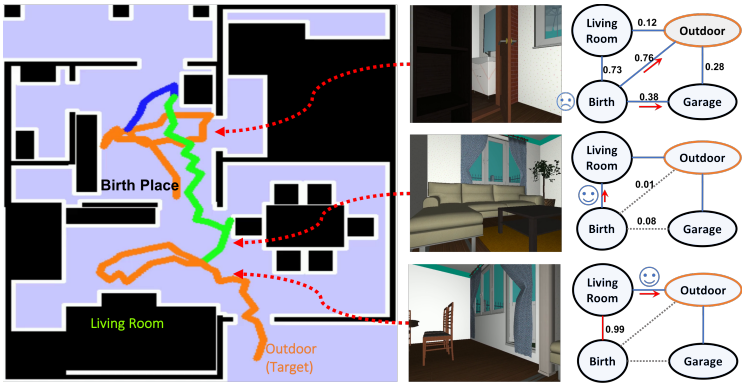

Fig. 3. Example of a successful trajectory. The agent is spawned inside the house, targeting the "outdoor". Left: the 2D top-down map with subtask trajectories ("outdoor" – orange; "garage" – blue; "living room" – green); Middle: RGB visual image; Right: the posterior of the connectivity graph and the proposed subtasks (red arrow). Initially, the agent starts by executing the sub-policy "outdoor" and then "garage" according to the prior knowledge, but both fail (top orange and blue trajectories). After updating its belief that garage and outdoor are not nearby (grey edges in graph), it then executes the "living room" sub-policy with success (red edges in graph, green trajectory). Finally, it executes "outdoor" sub-policy again, explores the living room and reaches the goal (bottom orange trajectory).

5.5 Extension: from RoomNav to ObjectNav

Here we consider extending our targets from room types to object categories, e.g., chair, dresser, etc. Here, the agent is asked to find an instance of a particular object type. We call this task *ObjectNav*. From House3D, we select $K_o = 15$ target types denoted by $O_1, \ldots, O_{K_o}$. More task details are in the supplementary.

Extending the graph representation: Since an object instance can appear in multiple rooms (e.g., a chair can appear in both dining room and living room), we introduce additional Bernoulli variables $z^o_{i,j}$ for every pair T_i and O_j denoting whether room T_i contains an instance of object type O_j. We use the same approach as in RoomNav to learn the parameters ψ.

Extending the semantic signal s_s: In addition to the existing K bits corresponding to rooms, we introduce another $K_o = 15$ bits where $s_s(K+i)$ denotes whether the agent can "see" an instance of object type O_i. Here we assume that the agent has a reasonably working object detector: $s_s(K+i)$ becomes 1 if at least 1% pixels of the current input image belong to object type O_i.

Guided navigation for objects: We train K_o additional policies $\mu(\theta_{K+i})$ for navigating towards object O_i. During exploration, we plan according to Eq. 3.

Results: We compare our guided navigation approach to (1) random policy and (2) pure subpolicy $\mu(\theta)$. The results are shown in Fig. 5 where our approach outperforms all these baselines.

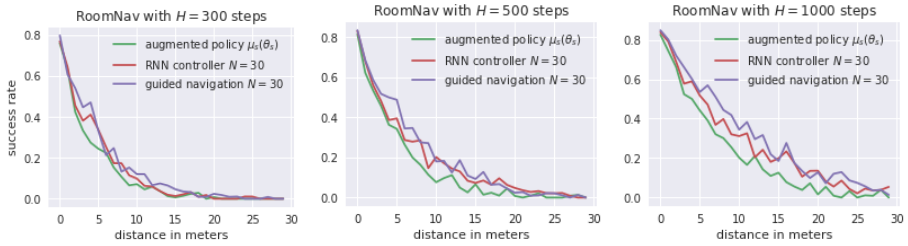

Fig. 4. Performance within different horizon H on RoomNav of augmented policy $\mu_B(\theta_B)$ (green), RNN controller with $N = 30$ (red) and our approach with graphical model M and $N = 30$ (purple). X-axis: distance from agent's birth place to target; Y-axis: success rate.

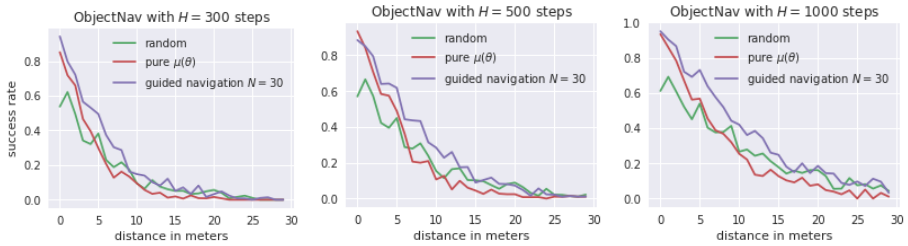

Fig. 5. Performance on ObjectNav within different horizon H of random policy (green), pure subpolicy $\mu(\theta)$ (red) and our guided navigation with $N = 30$ (purple). X-axis: distance from agent's birth place to target; Y-axis: success rate.

6 Conclusion

In this paper, we propose guided navigation by learning a graphical model over semantic properties that provides effective guidance to neural subpolicies even in completely unseen environments. Our approach is lightweight, interpretable and improves the success rate when the target is faraway. Our approach can be applied to general graphical models beyond our current simplified design. There are two potential directions for future work: (1) using semi-/non-parametric graphical models; (2) considering problems with deterministic relationships between targets.

References

1. Bacon, P.L., Harb, J., Precup, D.: The option-critic architecture. In: AAAI. pp. 1726–1734 (2017)
2. Duan, Y., Schulman, J., Chen, X., Bartlett, P.L., Sutskever, I., Abbeel, P.: Rl2: Fast reinforcement learning via slow reinforcement learning. arXiv preprint arXiv:1611.02779 (2016)
3. Finn, C., Abbeel, P., Levine, S.: Model-agnostic meta-learning for fast adaptation of deep networks. arXiv preprint arXiv:1703.03400 (2017)

4. Fox, D., Thrun, S., Burgard, W.: Probabilistic Robotics. MIT press (2005)
5. Gupta, S., Davidson, J., Levine, S., Sukthankar, R., Malik, J.: Cognitive mapping and planning for visual navigation. arXiv preprint arXiv:1702.03920 **3** (2017)
6. Hallak, A., Di Castro, D., Mannor, S.: Contextual markov decision processes. arXiv preprint arXiv:1502.02259 (2015)
7. Jaderberg, M., Mnih, V., Czarnecki, W.M., Schaul, T., Leibo, J.Z., Silver, D., Kavukcuoglu, K.: Reinforcement learning with unsupervised auxiliary tasks. arXiv preprint arXiv:1611.05397 (2016)
8. Kansky, K., Silver, T., Mély, D.A., Eldawy, M., Lázaro-Gredilla, M., Lou, X., Dorfman, N., Sidor, S., Phoenix, S., George, D.: Schema networks: Zero-shot transfer with a generative causal model of intuitive physics. arXiv preprint arXiv:1706.04317 (2017)
9. Khan, A., Zhang, C., Atanasov, N., Karydis, K., Kumar, V., Lee, D.D.: Memory augmented control networks. ICLR (2018)
10. Leonard, J.J., Durrant-Whyte, H.F.: Directed Sonar Sensing for Mobile Robot Navigation. Kluwer Academic Publishers, Norwell, MA, USA (1992)
11. Levine, S., Finn, C., Darrell, T., Abbeel, P.: End-to-end training of deep visuomotor policies. JMLR (2016)
12. Mirowski, P., Grimes, M.K., Malinowski, M., Hermann, K.M., Anderson, K., Teplyashin, D., Simonyan, K., Kavukcuoglu, K., Zisserman, A., Hadsell, R.: Learning to navigate in cities without a map. arXiv preprint arXiv:1804.00168 (2018)
13. Mirowski, P., Pascanu, R., Viola, F., Soyer, H., Ballard, A., Banino, A., Denil, M., Goroshin, R., Sifre, L., Kavukcuoglu, K., et al.: Learning to navigate in complex environments. arXiv preprint arXiv:1611.03673 (2016)
14. Mishra, N., Rohaninejad, M., Chen, X., Abbeel, P.: Meta-learning with temporal convolutions. arXiv preprint arXiv:1707.03141 (2017)
15. Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., Kavukcuoglu, K.: Asynchronous methods for deep reinforcement learning. In: International Conference on Machine Learning. pp. 1928–1937 (2016)
16. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529 (2015)
17. Oh, J., Singh, S., Lee, H., Kohli, P.: Zero-shot task generalization with multi-task deep reinforcement learning. arXiv preprint arXiv:1706.05064 (2017)
18. Parisotto, E., Salakhutdinov, R.: Neural map: Structured memory for deep reinforcement learning. arXiv preprint arXiv:1702.08360 (2017)
19. Riedmiller, M., Hafner, R., Lampe, T., Neunert, M., Degrave, J., Van de Wiele, T., Mnih, V., Heess, N., Springenberg, J.T.: Learning by playing-solving sparse reward tasks from scratch. arXiv preprint arXiv:1802.10567 (2018)
20. Savinov, N., Dosovitskiy, A., Koltun, V.: Semi-parametric topological memory for navigation. ICLR (2018)
21. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al.: Mastering the game of go with deep neural networks and tree search. nature **529**(7587), 484–489 (2016)
22. Song, S., Yu, F., Zeng, A., Chang, A.X., Savva, M., Funkhouser, T.: Semantic scene completion from a single depth image. IEEE Conference on Computer Vision and Pattern Recognition (2017)
23. Tamar, A., Wu, Y., Thomas, G., Levine, S., Abbeel, P.: Value iteration networks. In: Advances in Neural Information Processing Systems. pp. 2154–2162 (2016)

24. Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., Abbeel, P.: Domain randomization for transferring deep neural networks from simulation to the real world. arXiv preprint arXiv:1703.06907 (2017)
25. Vezhnevets, A.S., Osindero, S., Schaul, T., Heess, N., Jaderberg, M., Silver, D., Kavukcuoglu, K.: Feudal networks for hierarchical reinforcement learning. arXiv preprint arXiv:1703.01161 (2017)
26. Wu, Y., Wu, Y., Gkioxari, G., Tian, Y.: Building generalizable agents with a realistic and rich 3d environment. arXiv preprint arXiv:1801.02209 (2018)
27. Zhu, Y., Mottaghi, R., Kolve, E., Lim, J.J., Gupta, A., Fei-Fei, L., Farhadi, A.: Target-driven visual navigation in indoor scenes using deep reinforcement learning. In: Robotics and Automation (ICRA), 2017 IEEE International Conference on. pp. 3357–3364. IEEE (2017)

A Environment Details

In RoomNav the 8 targets are: kitchen, living room, dining room, bedroom, bathroom, office, garage and outdoor.

For ObjectNav, the 15 object targets are: kitchen cabinet, sofa, chair, toilet, table, sink, wardrobe cabinet, bed, shelving, desk, television, household appliance, dresser, vehicle and pool.

We inherit the success measure of "see" from [26]: the agent needs to see some corresponding object (in ObjectNav, it is the target object) for at least 450 pixels in the input frame and stay in the target area for at least 3 time steps.

For the binary signal s_s, we obtain from the bounding box information for each room provided from SUNCG [22].

Originally the House3D enviroment supports 13 discrete actions. Here we reduce it to 9 actions: large forward, forward, left-forward, right-forward, large left rotate, large right rotate, left rotate, right rotate and stay still.

B Details for Learning Neural Policies

We utilize the same policy architecture as [26]. We run A3C with $\gamma = 0.97$, batch size 64, learning rate 0.001 with Adam, weight decay 10^{-5}, entropy bonus 0.1. We backprop through at most 30 time steps. We also compute the squared l_2 norm of logits and added to the loss with a coefficient 0.01. We also normalize the advantage to mean 0 and standard deviation 1.

We run a curriculum learning by increasing the maximum of distance between agent's birth meters and target by 3 meters every 10000 iterations. We totally run 60000 training iterations and use the final model as our learned policy $\mu(\theta)$.

C Details for Learning Graphical Model

After evalution on the validation set, we choose to run random exploration for 300 steps to collect a sample of z. For a particular environment, we collect totally 50 samples for each $z_{i,j}$.

For all i, j, we set $\psi^y_{i,j,0} = 0.001$ and $\psi^y_{i,j,1} = 0.15$.

D Additional Experiment Details

For the RNN controller, we ran A2C with batch size 32, learning rate 0.001 with adam, weight decay 0.00001, gamma 0.99, entropy bonus 0.01 and advantage normalization. The reward function is designed as follows: for every subtask it propose, it gets a time penalty of 0.1; when the agent reach the target, it gets a success bonus of 2.

The input of RNN controller consists of (1) ${s_s}^{(t)}$ (K bits), (2) B (K bits), (3) last subtask T_k, and (4) the final target T_i. We convert T_i and T_k to a one-hot vector and combine the other two features to feed into the RNN. Hence the input dimension of RNN controller is $4K$, namely 32 in RoomNav.