BUILDING GENERALIZABLE AGENTS WITH A REALISTIC AND RICH 3D ENVIRONMENT

Anonymous authors
Paper under double-blind review

ABSTRACT

Towards bridging the gap between machine and human intelligence, it is of utmost importance to introduce environments that are visually realistic and rich in content. In this work, we build *House3D*, a rich, extensible and interactive environment that contains over 45,000 human-designed 3D scenes of houses, ranging from single-room studios to multi-storeyed houses, equipped with a diverse set of fully labeled 3D objects, textures and scene layouts, based on SUNCG dataset (Song et al., 2017). Using a subset of houses in this environment, we study the task of *RoomNav* in which an agent navigates towards a target specified by a high-level instruction. For this, the agent learns to comprehend the scene it lives in by developing perception, understand the instruction by mapping it to the correct semantics, and navigate to the target by obeying the underlying physical laws. We tackle this problem by training RL agents with gated-attention networks and show that our trained agent succeeds in new unseen environments, showing generalization capability. In particular, we observe that (1) training is substantially harder on large house sets but with better generalization, (2) using semantic signals (e.g., segmentation mask) boosts the generalization performance, and (3) gating mechanism helps more about the training but less about the generalization. We hope House3D, as well as the analysis of RoomNav task, acts as a step towards building a practical intelligent system and can potentially benefit the community.

1 INTRODUCTION

Recently, deep reinforcement learning has shown its strength on multiple games, such as Atari (Mnih et al., 2015) and Go (Silver et al., 2016), vastly overpowering human performance. Underlying these big achievements is a well-defined and efficient simulated environment for the agent to freely explore and learn. A natural question arises: will such simulated environments extrapolate to our real world? Till now, many proposed environments have encoded some aspects of human intelligence in the real-world. This includes 3D understanding (DeepMind Lab Beattie et al. (2016) and Malmo (Johnson et al., 2016)), real-time strategy decision (TorchCraft (Synnaeve et al., 2016) and ELF (Tian et al., 2017)), fast reaction (Atari (Bellemare et al., 2013)), long-term planning (Go, Chess), language and communications (ParlAI (Miller et al., 2017) and (Das et al., 2017)).

However, an intelligence system in real-world requires all these aspects in a comprehensive manner. For example (Fig. 1b), a workable home robot should be able to understand the perception, execute high-level instructions given by language, and explore and navigate with physical interactions, all at the same time. Hence, following the aforementioned trend, even if we made substantial progress on each of these environments, what we would get are specially skilled agents, but not almighty human-like intelligence. Although transfer learning can be used to transfer skills, a unified environment that could evaluate all such skills at once remains an open problem. Moreover, most of the environments miss an important aspect of real-world: the rich vision aspect. On average, humans can effortlessly recognize and detect 100k objects and deal with their complicated interactions. Unfortunately, many existing environments only contain a limited number of objects with compositional attributes.

In this paper, we propose House3D, a virtual 3D environment consisting of thousands of indoor scenes equipped with a diverse set of scene types, layouts and objects. House3D is adapted from the SUNCG dataset (Song et al., 2017) which contains 45K *human-designed* real-world 3D house models ranging from single studio to houses with gardens. We enhance the scenes with *physics* and

make it *interactive* so that an agent could explore the space freely, perceive with a large number of objects with various visual appearance, interact with them via simple physics, and even communicate with other agents via channels like language. These characteristics create rich visual textures in the observation space and complicated physical interactions, a step towards the real world.

House3D provides multiple modalities for training, including RGB image, ground truth segmentation masks, depth, 2D top-down map and various annotations of rooms and objects, which make it a very desirable choice for a wide range of research purposes. House3D is orders of magnitude larger and more diverse than other similar environments like AI2-THOR (Zhu et al., 2017) which contains only 30 single rooms for training and testing.

Among all the possible applications enabled by House3D, here we focus on a high-level instruction following task, RoomNav (Fig. 1b). In this task, an agent is randomly assigned at a location in the house and is asked to navigate to a destination described by a high-level concept (e.g., "kitchen") without colliding with objects. From the entire House3D, we manually select 270 houses that are suitable and large enough for navigation tasks, and split them into a small set and a large set for training and a test set for evaluation.

To train an agent on RoomNav, we propose two novel neural models with gated-attention architecture, a gated-CNN and a gated-LSTM, which effectively improve the agent's sensitivity to different instructions. Due to the complexity of House3D, we explore both continuous and discrete action space and apply two popular reinforcement learning algorithms, DDPG (Lillicrap et al., 2015) for continuous actions, and A3C (Mnih et al., 2016) for discrete actions.

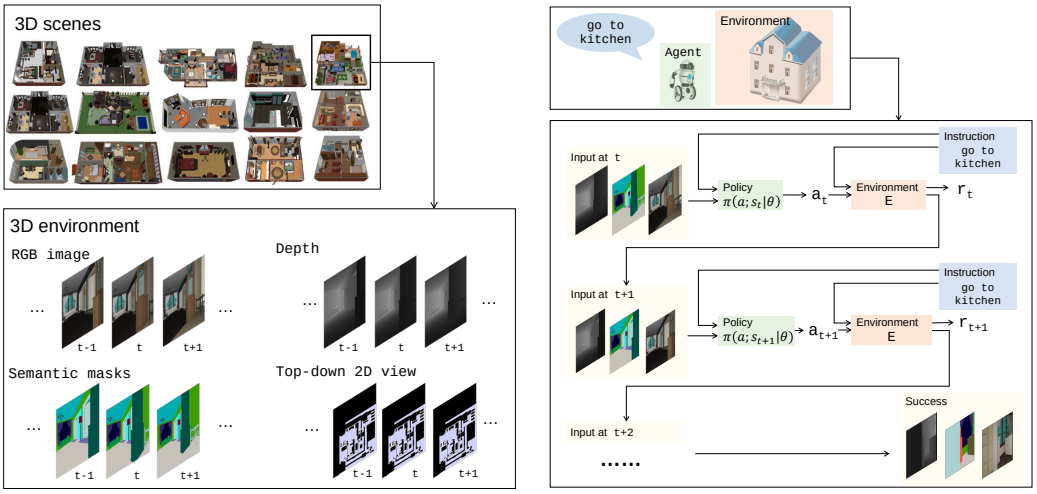
Further, when evaluating our model's performance, we emphasize on *generalizability*. Note that most of the existing reinforcement learning works focus on the performances on a fixed training environment. However, a well-trained intelligent robot for real-world applications should be successfully deployed to new scenarios. Indeed, there is a recent trend in the RL community focusing on the problem of generalization, ranging from learning to plan (Tamar et al., 2016), meta-learning (Duan et al., 2016; Finn et al., 2017) to zero-shot learning (Oh et al., 2017; Higgins et al., 2017). But these works either focus on simplified tasks or test environments which only slightly vary from the training ones. In this paper, we randomly split and preserve a held-out test set of houses on which we examine the agent's navigation success rate. All the environments in the test set are completely different from the training houses and are never presented to the agent during training.

In our experiments, we notice that a larger training set leads to higher generalization ability, which suggests the importance of divergence of simulated scenery. We also experiment with different visual inputs to the agent and conclude that the visual signal leading to immediate action (depth) or the semantic signal (segmentation) can lead to better generalization. This empirical observation indicates the significance of building a practical vision system for real-world robots and also suggests the separation of vision and robotic learning when handling complicated real-world tasks.

The remaining of the paper is structured as follows. Section 2 is a summary of relevant work. Section 3 describes our environment, House3D, in detail and Section 4 describes the task, RoomNav. Section 5 introduces our gated-networks and the applied algorithms. Finally, experiment results are shown in Section 6.

2 Related Work

The development of environments largely pushes the frontier of reinforcement learning. Table 1 summarizes the attributes of some of the most relevant environments and compares them to our House3D. Other than the family of environments closely related to House3D, there are more simulated environments, such as OpenAI Gym (Brockman et al., 2016), ParlAI (Miller et al., 2017) for language communication as well as some strategic game environments (Synnaeve et al., 2016; Tian et al., 2017; Vinyals et al., 2017). Most of these environments are specialized in one particular aspect of intelligence, such as game or dialogue, and make it hard to facilitate the study of more problems. On the contrary, we focus on building a flexible platform that intersects with multiple research directions in an efficient manner allowing users to customize the rules and level of complexity to their needs.



(a) House3D environment

(b) RoomNav task

Figure 1: An overview of House3D environment and RoomNav task. **(a)** We build an efficient and interactive environment upon the SUNCG dataset (Song et al., 2017) that contains 45K diverse indoor scenes, ranging from studios to two-storied houses with swimming pools and fitness rooms. All 3D Objects are fully annotated into over 80 categories. Agents in the environment have access to observations of multiple modalities (e.g., RGB images, Depth, Segmentation masks (from object category), top-down 2D view, etc. **(b)** We focus on the task of targeted navigation. Given a high-level task description, the agent explores the environment to reach the target room.

Environment	3D	Realistic	Large-scale	Fast-speed	Customizable
Atari (Bellemare et al., 2013)				●	
OpenAI Universe (Shi et al., 2017)		●	●		●
Malmo (Johnson et al., 2016)	●		●	●	●
DeepMind Lab (Beattie et al., 2016)	●			●	●
VizDoom (Kempka et al., 2016)	●			●	●
AI2-THOR (Zhu et al., 2017)	●	●		●	
House3D	●	●	●	●	●

Table 1: A summary of popular environments. The attributes include **3D**: 3D nature of the rendered objects, **Realistic**: resemblance to the real-world, **Large-Scale**: a large set of environments, **Fast-speed**: fast rendering speed and **Customizable**: flexibility to be customized to any desired applications.

There has been noticeable efforts to tackle navigation in 3D environments using reinforcement learning. Some of the most prominent lines of work include the use of auxiliary tasks to improve training speed (Mirowski et al., 2016; Jaderberg et al., 2016), the use of successor representations (Kulkarni et al., 2016), memory (Parisotto & Salakhutdinov, 2017) and self-supervision (Pathak et al., 2017) in VizDoom (Kempka et al., 2016).

In our work, the goal is not specified by the environment but is communicated to the agent via a high-level instruction. To modulate the behavior of the agent, we encode the instruction as an embedding vector which gates the visual signal. The idea of gated attention has been used in the past for language grounding (Chaplot et al., 2017), and transfer learning by language grounding (Narasimhan et al., 2017). Similar to those works, we use language grounding as an attention mechanism. However, the effects of the grounding are directed to impact the agent's overall perception and thus future actions towards the goal.

For stronger generalization performance and efficient learning, we rely on multi-task training. A similar strategy is followed by Tessler et al. (2016); Andreas et al. (2016); Oh et al. (2017); Teh et al. (2017). In addition and similar to other works which use diverse input modalities (Mirowski et al.,

2016; Tai & Liu, 2016), we show improved generalization performance in complex 3D scenes when using depth and segmentation masks on top of the raw visual input. This observation suggests that it is possible to decouple real-world robotics from vision via a vision API. The vision API trained in the desired scenes, e.g. an object detection or semantic segmentation system, can bridge the gap between simulated environment and real-world (Tobin et al., 2017; Rusu et al., 2016; Christiano et al., 2016).

3 HOUSE3D: AN EXTENSIBLE ENVIRONMENT OF 45K 3D HOUSES

Towards building an ultimately practical AI system, there is a need for a realistic environment rich in content and structure which closely resembles the real world. Such an environment can serve as the testbed for various problems which require visual understanding, language grounding, concept learning and more. For the environment to be of value, it is important that it obeys the physical laws of the world, carries a complex visual signal and is of large enough scale to enable learning and generalization. For this, we develop an environment of thousands of indoor scenes, an overview of which is shown in Figure 1a. An agent is able to navigate inside complex structures which consist of a diverse set of layouts, objects and rooms and are accompanied by useful semantic labels.

3.1 DATASET

The 3D scenes are sourced from the SUNCG dataset (Song et al., 2017), which consists of 45,622 human-designed 3D scenes ranging from single-room studios to multi-floor houses. The SUNCG dataset was designed to encourage research on large-scale 3D object recognition problems and thus carries a variety of objects, scene layouts and structures. On average, there is 8.9 rooms and 1.3 floors per scene with the max being 155 rooms and 3 floors, respectively. There is a diverse set of room and object types in each scene. In total, there is over 20 different room types, such as bedroom, living room, kitchen, bathroom etc., with over 80 different object categories. On average, there is 14 different objects in each room. In total, the SUNCG dataset contains 404,508 different rooms and 5,697,217 object instances drawn from 2644 unique object meshes.

3.2 ANNOTATIONS

SUNCG is accompanied with a diverse set of labels for each scene. Based on these labels, an agent at every time step t can have access to the following signals in our environment: a) the visual signal of its current first-person view, consisting of RGB values, b) semantic segmentation masks for all the objects visible in its current view and c) depth information. These signals are greatly desired since they enable thorough exploration to determine their practical value for different tasks or can serve as a predictive target when learning models. For example, for the task of navigation one can easily swap the RGB values with depth information or with semantic masks in order to quantify the importance of these different input signals.

Other than the visual input, each scene in SUNCG is fully annotated with the 3D location and size of all rooms and object instances in the form of a 3D bounding box. Rooms and objects are also marked with their type, e.g. bedroom or shoe cabinet respectively. This information allows for a detailed mapping from each 3D location (x, y, z) to an object instance (or None if the space is unoccupied), as well as room type. Furthermore, more features can be built on top of these existing annotations, such as top-down 2D occupancy maps, connectivity analysis and shortest paths, or any arbitrary physics, all potentially helpful for a variety of applications.

3.3 BUILDING AN INTERACTIVE ENVIRONMENT

3.3.1 RENDERER

To build a realistic 3D environment, we write a renderer for the 3D scenes from SUNCG, and define actions that obey simple physical rules for an agent to navigate in the scenes. The renderer is based on OpenGL and can run on Linux and macOS with both CPU and GPU. The renderer provides not only RGB images, but also segmentation masks and depth maps.

The renderer needs to be efficient to be used for large-scale RL algorithms. On a NVIDIA Tesla M40 GPU, it can render 120x90 RGB frames at over 600 fps, while multiple renderers can run in parallel on one or more GPUs. When rendering multiple houses simultaneously, one M40 GPU can be fully utilized to render at a total of 1800 fps. The simple physics, currently written in Python, add a small overhead to the compute, which is negligible when used with multi-processing. The high throughput of our implementation enables efficient learning for a variety of interactive tasks, such as on-policy reinforcement learning.

We will publicly release the environment along with a python API for easy use.

3.3.2 PHYSICS

For the environment to be realistic, we incorporate physical laws to the SUNCG scenes based on occupancy. Particularly, an agent can live in any location (x, y, z) within a 3D scene, as long as within a small range, i.e., robot's radius, it is not collided with any object instance (including walls). Doors, gates and arches are considered passage ways for the agent, meaning that an agent can walk through those structures freely. These design choices closely follow the behaviour of a real robot navigating inside a house: a robot can not walk through walls or other objects but can pass freely through free space, including doors and gates, in order to reach different parts of the scene.

4 ROOMNAV: A BENCHMARK TASK OF MULTI-TARGET ROOM NAVIGATION

Consider a natural use case of a home robot in Figure 1b. A human may give a high level instruction to the robot, for example, "Go to the kitchen", so that one can later ask the robot to turn on the oven. The robot needs to behave appropriately conditioned on the house it is located in as well as the instruction from the human. Moreover, a practical commercial home robot also needs the ability of *generalization*: the robot can be intensively trained for an arbitrarily long time by its manufacturer but it needs to be immediately deployed to the user's home, which can be of a completely different house design and indoor decorations from the training environments.

The key challenges for building real-world home robots are: (1) understanding of the scene from its visual sensors; (2) execution of high-level instructions; (3) safe navigation and exploration in complex indoor environments; (4) generalization to unseen scenes.

To study the aforementioned abilities of a reinforcement learning agent, we develop a new benchmark task, *Multi-Target Room Navigation (RoomNav)*, based on our House3D environment. To alleviate the difficulty of the task, we restrict the high-level instruction to be of the form "go to `RoomType`", where `RoomType` denotes some pre-defined room type. To ensure fast experimentation cycles, we perform experiments on a subset of House3D. We manually select 270 houses suitable for navigation task and split them into a small set (20 houses), a large set (200 houses) and a test set (50 houses), where the test set is used to evaluate the generalizability of the trained agents.

4.1 TASK FORMULATION

Suppose we have a set of episodic environments $\mathcal{E} = \{E_1, \ldots, E_n\}$ and a set of targeted instructions $\mathcal{I} = \{I_1, \ldots, I_m\}$. During each episode, the agent is interacting with one environment $E \in \mathcal{E}$ and is given a targeted instruction $I \in \mathcal{I}$. In the beginning of an episode, the agent will be randomly placed somewhere in E. At each time step t, the agent receives a visual signal X_t from E via its first person view sensor. Let $s_t = \{X_1, \ldots, X_t, I\}$ denote the state of the agent at time t. The agent needs to propose action a_t to navigate and rotate its sensor given s_t. The environment will give out a reward signal r_t and terminates when it succeeds or times out.

The objective of this task is to learn an optimal policy $\pi(a_t|s_t, I)$ that could lead to the targeted room according to I. During training, we train the agent in a training set of environments $\mathcal{E}_{\text{train}}$. For measuring the generalization of the learned policy, we will also evaluate the policy in another set of environments $\mathcal{E}_{\text{test}}$, such that $\mathcal{E}_{\text{test}} \cap \mathcal{E}_{\text{train}} = \emptyset$.

4.2 DETAILED SPECIFICATIONS

Here we provide further details of the task and there are more to find in appendix.

	$\lvert\mathcal{E}\rvert$	avg. #targets	kitchen%	dining room %	living room%	bedroom%	bathroom%
$\mathcal{E}_{\text{small}}$	20	3.9	0.95	0.60	0.60	0.95	0.80
$\mathcal{E}_{\text{large}}$	200	3.7	1.00	0.35	0.63	0.94	0.80
$\mathcal{E}_{\text{test}}$	50	3.7	1.00	0.48	0.58	0.94	0.70

Table 2: Statistics of the selected environment sets for RoomNav. `RoomType%` denotes the percentage of houses containing at least one target room of type `RoomType`.

Environment Statistics: The selected 270 houses are manually verified to be suitable for navigation task: e.g., they are well connected, contain many targets, and are large enough for exploration (studios excluded). We split them into 3 disjoint sets, denoted by $\mathcal{E}_{small}$, $\mathcal{E}_{large}$ and $\mathcal{E}_{test}$ respectively. For targets, we select the five most common room types, i.e., kitchen, living room, dining room, bedroom and bathroom. The detailed statistics are shown in Table 2.

Observation and Action: We utilize three different kinds of visual input signals for X_t, including (1) raw pixel values; (2) segmentation mask of the pixel input; and (3) depth information, and experiment with different combinations of them. For each instruction I, we provide a one-hot vector encoding correspondingly.

The majority of the existing navigation works operate on a discrete action space. In this work, we predefine 12 discrete actions including rotations and movement. In addition, due to the complexity of the indoor scenes as well as the fact that real robots may navigate towards any direction with any (bounded) velocity, we also investigate a continuous action space similar to Lowe et al. (2017). See the appendix for more details. For both the discrete and continuous action space, if the agent hits any obstacles, it will remain still.

Success Measure and Reward Function: To define "success" for the task, we want to ensure that the agent *identifies* the room due to its unique properties (e.g., pan and knives for kitchen and bed for bedroom) instead of merely *reaching* there by luck. An episode is considered successful if the agent achieves both of the following two criteria: (1) the agent is located inside the target room; (2) the agent consecutively "sees" a designated object category associated with that target room type for at least 2 time steps. We assume that an agent "sees" an object if there are at least 4% of pixels in X_t belonging to that object.

Regarding the reward function, ideally two signals are enough to reflect the task requirement: (1) a collision penalty when hitting obstacles; and (2) a success reward when completing the instruction. However, this basic reward function makes it too difficult for an RL agent to learn. Note that our ultimate goal is to learn a policy that could *generalize* to unseen houses at test time. Thus, during training we resort to an informative reward shaping in order to provide additional supervision: we compute the approximate shortest distance from the target room to each location in the house and adopt the difference of shortest distances between the agent's movement as an additional reward signal.

5 Gated-Attention Networks for Multi-Target Learning

In this section, we propose two baseline models with the gated-attention architecture, similar to Dhingra et al. (2016) and Chaplot et al. (2017), for better policy representation with instructions: a gated-CNN network for continuous actions and a gated-LSTM network for discrete actions. We train the gated-CNN policy using the deep deterministic policy gradient (DDPG) (Lillicrap et al., 2015), while we train the gated-LSTM policy using the asynchronous advantage actor-critic algorithm (A3C) (Mnih et al., 2016).

5.1 DDPG with gated-CNN policy

5.1.1 Deep deterministic policy gradient

DDPG is a widely used offline-learning algorithm for continuous action space (Lillicrap et al., 2015). Suppose we have a deterministic policy $\mu(s_t|\theta)$ (actor) and the Q-function $Q(s_t, a|\theta)$ (critic) both

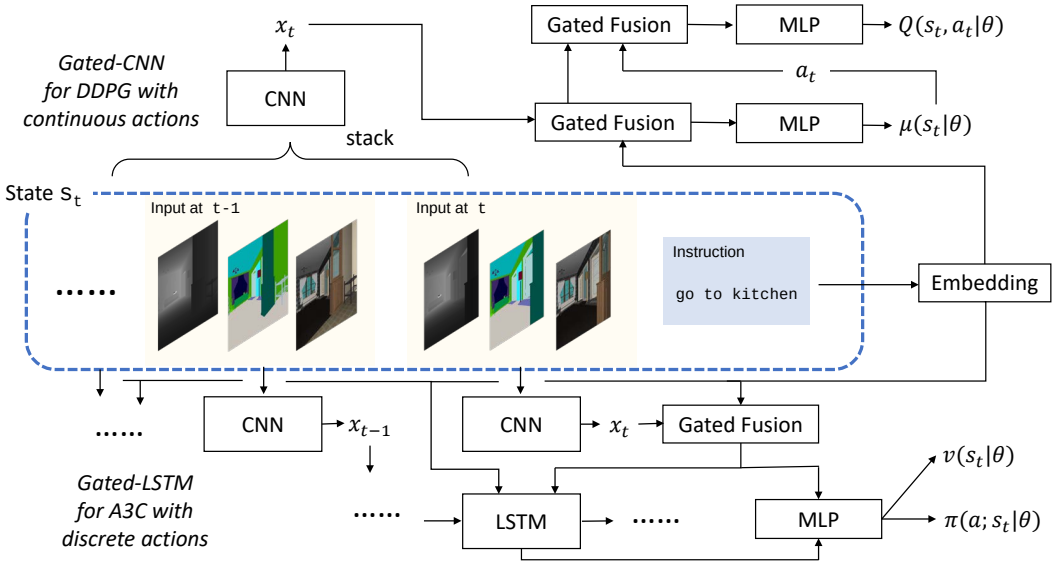


Figure 2: Overview of our proposed model. Bottom part demonstrates the gated-LSTM model for discrete action while the top part shows the gated-CNN model for continuous action. The "Gated Fusion" module denotes the gated-attention architecture.

parametrized by θ. DDPG optimizes the policy $\mu(s_t|\theta)$ by maximizing

$$L_\mu(\theta) = \mathbb{E}_{s_t}\left[Q(s_t, \mu(s_t|\theta)|\theta)\right],$$

and updates the Q-function by minimizing

$$L_Q(\theta) = \mathbb{E}_{s_t,a_t,r_t}\left[\left(Q(s_t, a_t|\theta) - \gamma Q(s_{t+1}, \mu(s_{t+1}|\theta)|\theta) - r_t\right)^2\right]$$

Here, we use a shared network for both actor and critic, which leads to the final loss function $L_{\text{DDPG}}(\theta) = -L_\mu(\theta) + \alpha_{\text{DDPG}} L_Q(\theta)$, where α_{DDPG} is a constant balancing the two objectives.

5.1.2 GATED-CNN NETWORK FOR CONTINUOUS POLICY

State Encoding: Given state s_t, we first channel-wise stack the most recent k frames $X = [X_t, X_{t-1}, \ldots, X_{t-k+1}]$ and apply a convolutional neural network to derive an image representation $x = f_{\text{cnn}}(X|\theta) \in \mathbb{R}^{d_X}$. For the instruction, we convert it into an embedding vector $y = f_{\text{embed}}(I|\theta) \in \mathbb{R}^{d_I}$. Subsequently, we apply a fusion module $M(x, y|\theta)$ to derive the final encoding $h_s = M(x, y|\theta)$.

Gated-Attention for Feature Fusion: For the fusion module $M(x, y|\theta)$, the straightforward version is concatenation, namely $M_{\text{cat}}(x, y|\cdot) = [x, y]$. In our case, x is always a high-dimensional feature vector (i.e., image feature) while y is a simple low-dimensional conditioning vector (e.g., instruction). Thus, simple concatenation may result in optimization difficulties. For this reason, we propose to use a gated-attention mechanism. Suppose $x \in \mathbb{R}^{d_x}$ and $y \in \mathbb{R}^{d_y}$ where $d_y < d_x$. First, we transform y to $y' \in \mathbb{R}^{d_x}$ via an MLP, namely $y' = f_{\text{mlp}}(y|\theta)$, and then perform a Hadamard product between x and $\text{sigmoid}(y')$, which leads to our final gated fusion module $M(x, y|\theta) = x \odot \text{sigmoid}(f_{\text{mlp}}(y|\theta))$. This gated fusion module could also be interpreted as an attention mechanism over the feature vector, which could help better shape the feature representation.

Policy Representation: On the state representation h_s, we apply an MLP layer followed by a softmax operator (for bounded velocity) to produce the continuous action. Moreover, in order to produce a stochastic policy for better exploration, we apply the softmax-gumbel trick (Jang et al., 2016), resulting in the final policy $\mu(s_t|\theta) = \text{gumbel-softmax}(f_{\text{mlp}}(h_s|\theta))$. Note that since we add randomness to $\mu(s_t|\theta)$, our DDPG formulation can be also interpreted as the SVG(0) algorithm (Heess et al., 2015).

Q-function: The Q-function $Q(s, a)$ conditions on both state s and action a. We again apply a gated fusion module to the feature vector x and the action vector a and derive a hidden representation $h_Q = M(x, a|\theta)$. We eventually apply another MLP to h_Q to produce the final value $Q(s, a)$.

A model demonstration is shown in the top part of Fig. 2, where each block has its own parameters.

5.2 A3C WITH GATED-LSTM POLICY

5.2.1 ASYNCHRONOUS ADVANTAGE ACTOR-CRITIC

A3C is a variant of policy gradient algorithm introduced by Mnih et al. (2016), which reduces the variance of policy gradient by jointly updating a value function as the baseline. Suppose we have a discrete policy $\pi(a; s|\theta)$ and a value function $v(s|\theta)$. A3C optimizes the policy by minimizing the loss function

$$L_{\text{pg}}(\theta) = -\mathbb{E}_{s_t, a_t, r_t}\left[\sum_{t=1}^{T}(R_t - v(s_t)) \log \pi(a_t; s_t|\theta)\right],$$

where R_t is the discounted accumulative reward defined by $R_t = \sum_{i=0}^{T-t} \gamma^i r_{t+i} + v(s_{T+1})$.

The value function is updated by minimizing the loss

$$L_v(\theta) = \mathbb{E}_{s_t, r_t}\left[(R_t - v(s_t))^2\right].$$

Finally the overall loss function for A3C is

$$L_{\text{A3C}} = L_{\text{pg}}(\theta) + \alpha_{\text{A3C}} L_v(\theta)$$

where α_{A3C} is a constant coefficient.

5.2.2 GATED-LSTM NETWORK FOR DISCRETE POLICY

State Encoding: Given state s_t, we first apply a CNN module to extract image feature x_t for each input frame X_t. Similarly, for instruction, we apply a gated fusion module to derive a state representation $h_t = M(x_t, I|\theta)$ at each time step t. Then, we concatenate h_t with the instruction I and the result is fed into the LSTM module (Hochreiter & Schmidhuber, 1997) to obtain a sequence of LSTM outputs $\{o_t\}_t$, so that the LSTM module has direct access to the instruction other than the attended visual feature.

Policy and Value Function: For each time step t, we concatenate the state vector h_t with the output of LSTM o_t to obtain a joint hidden vector $h_{\text{joint}} = [h_t, o_t]$. Then we apply two MLPs to h_{joint} to obtain the policy distribution $\pi(a; s_t|\theta)$ as well as the value function $v(s_t|\theta)$.

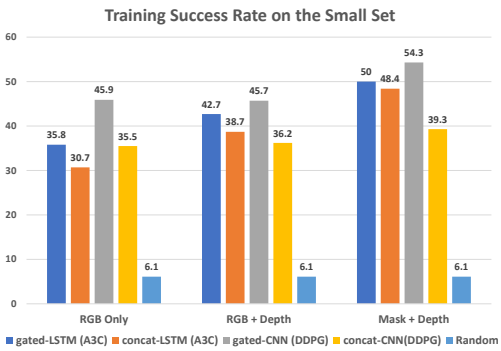
A visualization of the model is shown in the bottom part of Fig. 2. Note that the parameters of CNN module are shared across time.

6 EXPERIMENT RESULTS

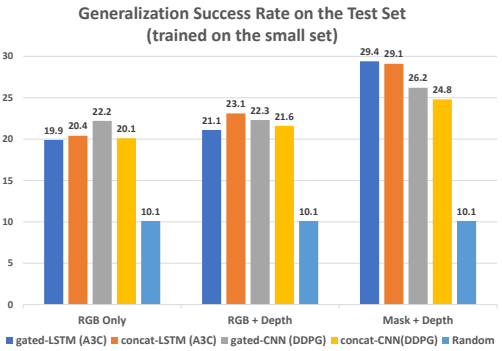
We report experimental results for our models on the task of RoomNav under different sized training sets. We compare models with discrete and continuous action spaces and empirically show the effect of using different input modalities.

6.1 DETAILS

We train our CNN model with DDPG and LSTM model with A3C on both the small set ($\mathcal{E}_{\text{small}}$, 20 houses) and the large set ($\mathcal{E}_{\text{large}}$, 200 houses), and examine both the training performances (success rate) on the training set as well as the generalization performances on the test set ($\mathcal{E}_{\text{test}}$, 50 houses). All the success rate evaluation uses a fixed random seed for a fair comparison. In all the cases, if the agent cannot accomplish the goal within 100 steps, we terminate the episode and declare failure. We use gated-CNN and gated-LSTM to denote the networks with gated-attention and concat-CNN and concat-LSTM for models with simple concatenation. We also experiment with different visual signals to the agents, including RGB image (RGB Only), RGB image with depth information



(a) Training performances



(b) Generalization performances on the test set

Figure 3: Performances of various models trained on $\mathcal{E}_{\text{small}}$ (20 houses) with different input signals: RGB Only, RGB+Depth and Mask+Depth. In each group, the bars from left to right correspond to gated-LSTM, concat-LSTM, gated-CNN, concat-CNN and random policy respectively.

(RGB+Depth) and semantics mask with depth information (Mask+Depth). To preserve the richness of the input visual signal, the resolution of the input image is 120×90.

During each simulated episode, we randomly select a house from the environment set and randomly pick an available target from the house to instruct the agent. Each episode will be forced to terminate after 100 time steps. During training, we add an entropy bonus term for both models in addition to the original loss function. For evaluation, we keep the final model for DDPG due to its stable learning curve, while for A3C, we take the model with the highest training success rate. Optimization is performed via Adam (Kingma & Ba, 2014) while the implementation is done in PyTorch (Paszke et al., 2017). More model and experiment details can be found in appendix.

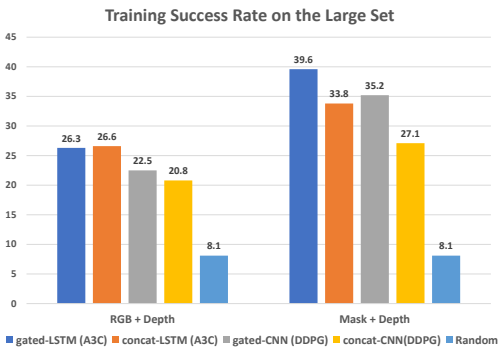
6.2 Training on the small set with 20 houses

Training Performances: For each of the models, we run 2000 evaluation episodes on $\mathcal{E}_{\text{small}}$ and measure overall the success rate. The results are summarized in Fig. 3a. Our empirical results show that the gated models achieve higher success rate than the models without gated-attention architecture. Especially for the DDPG case, the gated-CNN model outperforms the concat-CNN model with a large margin. We believe this is due to the fact that there are two gated-fusion modules utilized in gated-CNN model. Notably, the simpler CNN model with DDPG has stable learning curve and even achieves higher success rate than LSTM with all different input signals, which suggests that simpler CNN models can be an appropriate candidate for fitting a small set of environments.

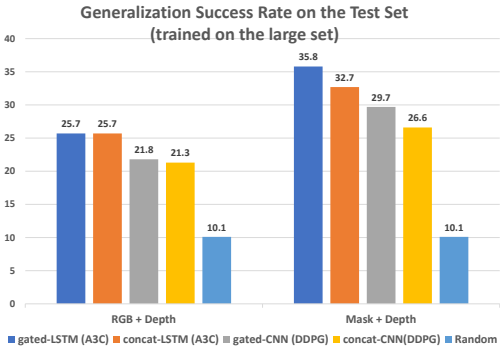
Regarding the different combinations of input signals, adding a depth channel to the RGB channel generally improves the training performances. Furthermore, changing the RGB signal to semantic signal significantly boosts the performances for all the models.

Generalization Performances on $\mathcal{E}_{\text{test}}$: Here, the models trained on $\mathcal{E}_{\text{small}}$ are evaluated on the test set $\mathcal{E}_{\text{test}}$ to measure their generalization ability. We run 2000 episodes and measure the success rate of each model. The results are shown in Fig. 3b.

Regarding different types of visual signals, we draw the same conclusions as at training time: depth improves the performances and semantic information contributes to generalizability the most. More importantly, we observe that the generalization performance is drastically worse than the training performance, especially for the cases with RGB input: the gated-LSTM models achieve even lower success rate than concat-LSTM models at test time despite the fact that they have much better training performances. This indicates that the neural model tends to *overfit* to the environments when we only have a small training set, while having a semantic input signal could slightly alleviate the issue of overfitting.



(a) Training performances



(b) Generalization performances on the test set

Figure 4: Performances of various models trained on $\mathcal{E}_{\text{large}}$ (200 houses) with input signals of RGB+Depth and Mask+Depth. In each group, the bars from left to right correspond to gated-LSTM, concat-LSTM, gated-CNN, concat-CNN and random policy respectively.

6.3 Training on the large set with 200 houses

Here we train our models on the large set $\mathcal{E}_{\text{large}}$ containing 200 different houses. For visual signals, we focus on "RGB + Depth" and "Mask + Depth". For evaluation, we measure the training success rate over 5000 evaluation episodes and the test success rate over 2000 episodes. Both train and test results are summarized in Fig. 4.

Training Performances: Compared to the performance on the small training set, the training performances on the large set drops significantly, especially for the models using RGB signal. We also notice that in the case of RGB input, gated models perform similar to concat-models. This suggests fundamental difficulties for reinforcement learning agents to learn within a *large* set of diverse and visually rich environments. Whereas, for models with semantic signals, we observe a huge gain on training performances as well as the benefits by having a gated-attention architecture. This implies that a semantic vision system can be potentially an effective component for building a real-world robotic system.

In addition, we observe that on the large set, even with a relatively unstable algorithm A3C, the models with larger capacity, i.e. LSTMs, considerably outperform the simpler reactive models, i.e., CNNs. We believe this is due to the larger scale and the high complexity of the training set, which makes it almost impossible for an agent to "remember" the optimal actions for every scenario. Instead, an agent needs to develop high-level abstract strategies, such as exploration, and memory. This also suggest a potential future direction of persuading the agent to learn generalizable abstractions by introducing more inductive bias into the model.

Generalization Performances: Regarding generalization, as we can see from Fig. 4b, after training on a large number of environments, every model now has a much smaller gap between its training performance and test performance. For the models with semantic input signals, their test performances are largely improved compared to those trained on the small set. This again emphasizes the importance of having a large set of diverse environments for training generalizable agents.

Lastly, we also analyze the detailed success rate with respect to each target instruction in appendix.

7 Conclusion

In this paper, we design a new environment, House3D, which contains 45K of houses with a rich set of objects as well as natural layouts resembling the real-world. House3D is a flexible and extensible environment that could be used for a variety of applications.

Based on our House3D environment, we further introduce a multi-target navigation task, RoomNav, which tests an agent's intelligence including understanding the instruction, interpreting the comprehensive visual signal, navigation and, most importantly, generalization.

We develop two novel models, gated-CNN and gated-LSTM, with the gated-attention architecture, which significantly helps improve the training performances.

In our experiments, we observe that using the semantic signal as the input considerably enhances the agent's generalizability. Increasing the size of the training environments is important but at the same time introduces fundamental bottlenecks when training agents to accomplish the RoomNav task due to the higher complexity of the underlying task.

We believe our environment will benefit the community and facilitate the efforts towards building better AI agents. We also hope that our initial attempts towards addressing generalizability in reinforcement learning will serve as an important step towards building real-world robotic systems.

REFERENCES

Jacob Andreas, Dan Klein, and Sergey Levine. Modular multitask reinforcement learning with policy sketches. *arXiv preprint arXiv:1611.01796*, 2016.

Charles Beattie, Joel Z Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, et al. Deepmind lab. *arXiv preprint arXiv:1612.03801*, 2016.

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *J. Artif. Intell. Res.(JAIR)*, 47:253–279, 2013.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Devendra Singh Chaplot, Kanthashree Mysore Sathyendra, Rama Kumar Pasumarthi, Dheeraj Rajagopal, and Ruslan Salakhutdinov. Gated-attention architectures for task-oriented language grounding. *arXiv preprint arXiv:1706.07230*, 2017.

Paul Christiano, Zain Shah, Igor Mordatch, Jonas Schneider, Trevor Blackwell, Joshua Tobin, Pieter Abbeel, and Wojciech Zaremba. Transfer from simulation to real world through learning deep inverse dynamics model. *arXiv preprint arXiv:1610.03518*, 2016.

Abhishek Das, Satwik Kottur, Stefan Lee, Jos M.F. Moura, and Dhruv Batra. Learning cooperative visual dialog agents with deep reinforcement learning. *ICCV*, 2017.

Bhuwan Dhingra, Hanxiao Liu, Zhilin Yang, William W Cohen, and Ruslan Salakhutdinov. Gated-attention readers for text comprehension. *arXiv preprint arXiv:1606.01549*, 2016.

Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, and Pieter Abbeel. RL^2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400*, 2017.

Nicolas Heess, Gregory Wayne, David Silver, Tim Lillicrap, Tom Erez, and Yuval Tassa. Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems*, pp. 2944–2952, 2015.

Irina Higgins, Arka Pal, Andrei A Rusu, Loic Matthey, Christopher P Burgess, Alexander Pritzel, Matthew Botvinick, Charles Blundell, and Alexander Lerchner. Darla: Improving zero-shot transfer in reinforcement learning. *arXiv preprint arXiv:1707.08475*, 2017.

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

Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Matthew Johnson, Katja Hofmann, Tim Hutton, and David Bignell. The malmo platform for artificial intelligence experimentation. In *IJCAI*, pp. 4246–4247, 2016.

Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. Vizdoom: A doom-based ai research platform for visual reinforcement learning. In *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*, pp. 1–8. IEEE, 2016.

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

Tejas D. Kulkarni, Ardavan Saeedi, Simanta Gautam, and Samuel J. Gershman. Deep successor reinforcement learning. *arXiv preprint arXiv:1606.02396*, 2016.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *arXiv preprint arXiv:1706.02275*, 2017.

Alexander H Miller, Will Feng, Adam Fisch, Jiasen Lu, Dhruv Batra, Antoine Bordes, Devi Parikh, and Jason Weston. Parlai: A dialog research software platform. *arXiv preprint arXiv:1705.06476*, 2017.

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

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pp. 1928–1937, 2016.

Karthik Narasimhan, Regina Barzilay, and Tommi Jaakkola. Deep transfer in reinforcement learning by language grounding. *arXiv preprint arXiv:1708.00133*, 2017.

Junhyuk Oh, Satinder Singh, Honglak Lee, and Pushmeet Kohli. Zero-shot task generalization with multi-task deep reinforcement learning. *arXiv preprint arXiv:1706.05064*, 2017.

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

Adam Paszke, Sam Gross, and Soumith Chintala. Pytorch, 2017. URL http://pytorch.org/.

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

Andrei A Rusu, Matej Vecerik, Thomas Rothörl, Nicolas Heess, Razvan Pascanu, and Raia Hadsell. Sim-to-real robot learning from pixels with progressive nets. *arXiv preprint arXiv:1610.04286*, 2016.

Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. World of bits: An open-domain platform for web-based agents. In *International Conference on Machine Learning*, pp. 3135–3144, 2017.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Shuran Song, Fisher Yu, Andy Zeng, Angel X Chang, Manolis Savva, and Thomas Funkhouser. Semantic scene completion from a single depth image. *CVPR*, 2017.

Gabriel Synnaeve, Nantas Nardelli, Alex Auvolat, Soumith Chintala, Timothée Lacroix, Zeming Lin, Florian Richoux, and Nicolas Usunier. Torchcraft: a library for machine learning research on real-time strategy games. *arXiv preprint arXiv:1611.00625*, 2016.

Lei Tai and Ming Liu. Towards cognitive exploration through deep reinforcement learningfor mobile robots. *arXiv preprint arXiv:1610.01733*, 2016.

Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value iteration networks. In *Advances in Neural Information Processing Systems*, pp. 2154–2162, 2016.

Yee Whye Teh, Victor Bapst, Wojciech Marian Czarnecki, John Quan, James Kirkpatrick, Raia Hadsell, Nicolas Heess, and Razvan Pascanu. Distral: Robust multitask reinforcement learning. *arXiv preprint arXiv:1707.04175*, 2017.

Chen Tessler, Shahar Givnoy, Tom Zahavy, Daniel J. Mankowitz, and Shie Mannor. A deep hierarchical approach to lifelong learning in minecraft. *arXiv preprint arXiv:1604.07255*, 2016.

Yuandong Tian, Qucheng Gong, Wenling Shang, Yuxin Wu, and Larry Zitnick. Elf: An extensive, lightweight and flexible research platform for real-time strategy games. *arXiv preprint arXiv:1707.01067*, 2017.

Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. *arXiv preprint arXiv:1703.06907*, 2017.

Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, et al. Starcraft ii: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*, 2017.

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 *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pp. 3357–3364. IEEE, 2017.

A ROOMNAV TASK DETAILS

The location information of an agent can be represented by 4 real numbers: the 3D location (x, y, z) and the rotation degree ρ of its first person view sensor, which indicates the front direction of the agent. Note that in RoomNav, the agent is not allowed to change its height z, hence the overall degree of freedom is 3.

An action can be in the form of a triple $a = (\delta_x, \delta_y, \delta_\rho)$. After taking the action a, the agent will move to a new 3D location $(x + \delta_x, y + \delta_y, z)$ with a new rotation $\rho + \delta_\rho$. The physics in House3D will detect collisions with objects under action a and in RoomNav, the agent will remain still in case of a collision. We also restrict the velocity of the agent such that $|\delta_x|, |\delta_y| \leq 0.5$ and $|\delta_\rho| \leq 30$ to ensure a smooth movement.

Continuous Action: A continuous action a consists of two parts $a = [m, r]$ where $m = (m_1, \ldots, m_4)$ is for movement and $r = (r_1, r_2)$ is for rotation. Since the velocity of the agent should be bounded, we require m, r to be a valid probability distribution. Suppose the original location of robot is (x, y, z) and the angle of camera is ρ, then after executing a, the new 3D location will be $(x + (m_1 - m_2) * 0.5, y + (m_3 - m_4) * 0.5, z)$ and the new angle is $\rho + (r_1 - r_2) * 30$.

Discrete Actions: We define 12 different action triples in the form of $a_i = (\delta_x, \delta_y, \delta_\rho)$ satisfying the velocity constraints. There are 8 actions for movement and 4 actions for rotation. In the discrete action setting, we do not allow the agent to move and rotate simultaneously.

Reward Details: In addition to the reward shaping of difference of shortest distances, we have the following rewards. When hitting an obstacle, the agent receives a penalty of 0.3. In the case of success, the winning reward is +10. In order to encourage exploration (or to prevent eternal rotation), we add a time penalty of 0.1 to the agent for each time step outside the target room. Note that since we restrict the velocity of the agent, the difference of shortest path after an action will be no more than $0.5 \times \sqrt{2} \approx 0.7$.

B EXPERIMENT DETAILS

B.1 NETWORK ARCHITECTURES

We apply a batch normalization layer after each layer in the CNN module. The activation function used is ReLU. The embedding dimension of instruction is 25.

Gated-CNN: In the CNN part, we have 4 convolution layers of 64, 64, 128, 128 channels perspective and with kernel size 5 and stride 2, as well as a fully-connected layer of 512 units. We use a linear layer to transform the instruction embedding to a 512-dimension vector for gated fusion. The MLP for policy has two hidden layers of 128 and 64 units, and the MLP for Q-function has a single hidden layer of 64 units.

Gated-LSTM: In the CNN module, we have 4 convolution layers of 64, 64, 128, 128 channels each and with kernel size 5 and stride 2, as well as a fully-connected layer of 256 units. We use a linear layer to convert the instruction embedding to a 256-dimension vector. The LSTM module has 256 hidden dimensions. The MLP module for policy contains two layers of 128 and 64 hidden units, and the MLP for value function has two hidden layers of 64 and 32 units.

B.2 TRAINING PARAMETERS

We normalize each channel of the input frame to $[0, 1]$ before feeding it into the neural network. Each of the training procedures includes a weight decay of 10^{-5} and a discounted factor $\gamma = 0.95$.

DDPG: We stack $k = 5$ recent frames and use learning rate 10^4 with batch size 128. We choose $\alpha_{\text{DDPG}} = 100$ for all the settings except for the case with input signal of "RGB+Depth" on $\mathcal{E}_{\text{large}}$, where we choose $\alpha_{\text{DDPG}} = 10$. We use an entropy bonus term with coefficient 0.001 on $\mathcal{E}_{\text{small}}$ and 0.01 on $\mathcal{E}_{\text{large}}$. We use exponential average to update the target network with rate 0.001. A training update is performed every 10 time steps. The replay buffer size is 7×10^5. We run training for 80000 episodes in all. We use a linear exploration strategy in the first 30000 episodes.

A3C: We clip the reward to the range $[-1, 1]$ and use a learning rate $1e-3$ with batch size 64. We launch 120 processes on $\mathcal{E}_{\text{small}}$ and 200 on $\mathcal{E}_{\text{large}}$. During training we estimate the discounted accumulative rewards and back-propagate through time for every 30 time steps unrolled. We perform a gradient clipping of 1.0 and decay the learning rate by a factor of 1.5 when the difference of KL-divergence becomes larger than 0.01. For training on $\mathcal{E}_{\text{small}}$, we use a entropy bonus term with coefficient 0.1; while on $\mathcal{E}_{\text{large}}$, the coefficient is 0.05. α_{A3C} is 1.0. We perform 10^5 training updates and keep the best model with the highest training success rate.

B.3 GENERALIZATION OVER DIFFERENT TARGETS

	test succ.	kitchen%	dining room %	living room%	bedroom%	bathroom%
gated-LSTM	35.8	37.9	50.4	48.0	33.5	21.2
gated-CNN	29.7	31.6	42.5	54.3	27.6	17.4

Table 3: Detailed test success rates for gated-CNN model and gated-LSTM model with "Mask+Depth" as input signal across different instructions.

We illustrate in Table 3 the detailed test success rates of our models trained on $\mathcal{E}_{\text{train}}$ with respect to each of the 5 targets. Note that both models have similar behaviour across instructions. In particular, "dining room" and "living room" are the easiest while "bathroom" is the hardest. We suspect that this is because dining room and living room are often with large room space and have the best connectivity to other places. By contrast, bathroom is often very small and harder to find in big houses.

Lastly, we also experiment with adding auxiliary tasks of predicting the current room type during training. We found this does not help the training performance nor the test performance. We believe it is because our reward shaping has already provided strong supervision signals.