# Learning by Asking Questions

Chaonan Song

Jun 20, 2018

## Abstract

*Today I read a part of the back of the article. In this section, the Abhishek team gave a more in-depth description of the environment in which their agents are located and the questions that the agents must answer. Then introduced the neural architecture that introduced their EmburalQA.*

## 1. EQA Dataset: Questions In Environments

In this section, they introduced in more depth the environment in which their agents are located and the issues they must answer. And publicly publish their EQA data set and data set generation code.

### 1.1. House3D: Simulated 3D Environments

The Abhishek team instantiated EmbodiedQA in House3D, a rich simulation environment for a recently launched 3D indoor scene based on the SUNG dataset Song *et al.* [4]. The reality of each scene was further verified by a majority vote of three human annotators. In general, SUNCG contains an environment of more than 45k, has 49k valid floors, and 404k rooms contain 5 million object instances from 80 different categories of 2644 unique objects. House3D converts SUNG from a static 3D data set into a set of simulation environments, and the agent can navigate under simple physical constraints. The figure 1shows a top view of the sample environment. Abhishek team builds EQA on House3D's pruning environment subset. First, the Abhishek team only considers the circumstances in which the three SUNGC annotators think the scene layout is real. Next, they screen for atypical environments, such as those that lack "ground" or those that are too small/too large. Finally, they need a home containing at least a kitchen, living room, dining room and bedroom.

Figure 1. Sample environments.

### 1.2. Question Answer Generation

The Abhishek team wants to ask questions to test the agent's language skills, use common sense, visual reasons, and the ability to navigate invisible environments. For example, answer "What color is the car?"Need to ground the symbol 'car', infer that the car is usually outside, navigate outside, explore the car and visually inspect its color.

Abhishek team draw inspiration from the CLEVR Johnson *et al.* [2] dataset and programmatically generate a dataset (EQA) with basic questions and answers. This allows us to carefully control the distribution of problem types and answers in the data set and prevent algorithms that use data set bias.

**Queryable Rooms and Objects.** Figures 2, 3 show 12 queryable rooms and 50 objects in the E-QA. The Abhishek team excludes objects or rooms that are obscure or difficult to understand intuitively from SUNCG. The Abhishek team incorporates semantically similar object categories, singular and plural, to reduce ambiguity.

**Questions as Functional Programs.** Each EQA problem is expressed as a functional program that can be executed in the environment to generate an answer. These programs consist of a set of basic operations that can operate on multiple sets of room or object comments and can be combined in any combination. The


Figure 2. Queryable rooms

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| rug | piano | dryer | computer | fireplace | whiteboard | bookshelf | wardrobe cabinet |
| pan | toilet | plates | ottoman | fish tank | dishwasher | microwave | water dispenser |
| bed | table | mirror | tv stand | stereo set | chessboard | playstation | vacuum cleaner |
| cup | xbox | heater | bathtub | shoe rack | range oven | refrigerator | coffee machine |
| sink | sofa | kettle | dresser | knife rack | towel rack | loudspeaker | utensil holder |
| desk | vase | shower | washer | fruit bowl | television | dressing table | cutting board |
| ironing board | | food processor | | | | | |

Figure 3. Queryable objects

number of these basic operations and the evaluation order define the problem type or template.

**Question Types.** In general, the Abhishek team has the following question types and associated templates in the EQA dataset: These questions test a variety of proxy capabilities, including object detection, scene recognition, counting, spatial reasoning, color identification, and logic. In addition, many of these problems require multiple skills - for example, answering distance problems requires identifying rooms and objects and inferring their spatial relationships. In addition, to do this, the agent must navigate the environment to find the room, then look around to find the object and remember their location.

**Question-Answer Generation and Dataset Bias.** In theory, they can now automatically generate all valid questions and related answers by executing functional programs on the environmental annotations provided by SUNCG. However, careful consideration must be given to ensuring that the data set developed is balanced on the answer.

**EQA v1 Statistics.** The EQA v1 dataset contains more than 5,000 questions in more than 750 environments, involving a total of 45 unique objects out of 7 unique room types. The data set is divided into train, val, test so that there is no overlap in the split environment. Figure 4 shows the data set split and problem type distribution. There are about 6 problems per environment, with a maximum of 22 and a minimum of 1. Since many frequently occurring spatial relationships are too easy to solve without exploration and cannot be entropy-thresholded, there are relatively few prepositional problems. We will announce the EQA v1 and the entire issue generation engine.

| | Environments | Unique Questions | Total Questions |
|---|---|---|---|
| train | 643 | 147 | 4246 |
| val | 67 | 104 | 506 |
| test | 57 | 105 | 529 |

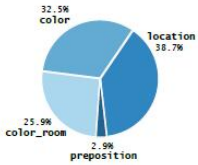


Figure 4. Overview of the EQA v1 dataset including dataset split statistics (left) and question type breakdown (right).

## 2. A Hierarchical Model for EmbodiedQA

In this section, Abhishek team introduced the neural architecture of their EmburalQA. Agents are generated at random locations in the environment, receive a problem, and can only be perceived by an ego-centric RGB camera. Importantly, unlike some previous work Andreas *et al.* [1], in EmbitureQA, the agent did not receive any environmental or structured environmental representations or tasks. **Overview of the Agent.** The agent has 4 natural modules - vision, language, navigation, answer - and provides training for visual questions from raw sensory input (pixel and text) to multi-room indoor navigation driven by goals. The modules themselves are mainly composed of traditional neural building blocks - Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN). The navigation module proposed by the Abhishek team is inspired by Graves's idea of adaptive computational time (ACT) RNN, which allows the RNN to learn how many computational steps to perform between receiving input and sending output. In their "planner-controller" navigation module (PACMAN), they use this idea to clearly distinguish between direction and speed. PACMAN is shown in Figure 5.

**Vision.** Their agent takes the self-centered 224224 RGB image of the House3D renderer as input. They use a CNN composed of four t55 Conv, BatchNorm, ReLU, and 22 MaxPoolu blocks to generate a fixed-size representation.

**Language.** The problem is encoded using a 2-layer LSTM with a 128-d hidden state. Although LSTM

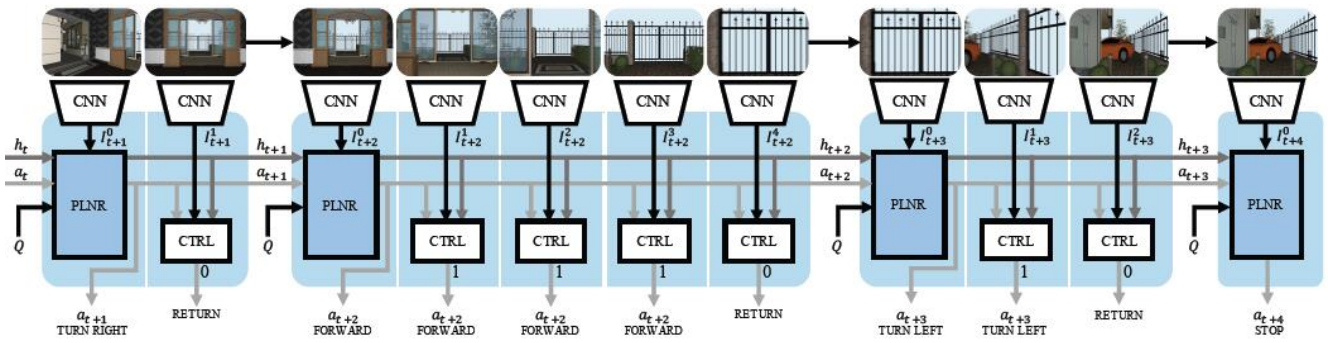


Figure 5. Their PACMAN navigator decomposes navigation into a planner and a controller. The planner selects actions and the controller executes these actions a variable number of times. This enables the planner to operate on shorter timescales, strengthening gradient flows.

may be overkill for simple problems in EQA v1, it can be flexibly extended to human-written questions or future more complex problem templates.

**Navigation.** Their PACMAN Navigator breaks down the navigation into "planners" and chooses to act before returning control to the planner. Intuitively, this structure separates the agent's intentions from the series of primitive actions required to implement this instruction, and is reminiscent of hierarchical RL. Formally, let $t = 1, 2, \ldots, T$ denote planner timesteps, and $n = 0, 1, 2, \ldots N(t)$ denote the variable number of controller timesteps. Let $I_t^n$ denote the encoding of the observed image at $t$-th planner-time and $n$-th controller-time. The planner is instantiated as an LSTM. Thus, it maintains a hidden state $h^t$ (updated only at planner timesteps), and samples action $a_t \in \{forward, turn-left, turn-right, stop\}$: Eq. 1

$$a_t, h_t \leftarrow PLNR(h_{t-1}, I_t^0, Q, a_{t-1}) \quad (1)$$

Where Q is the problem code. After taking this action, the planner passes control to the controller. The controller considers the planner's status and the current framework to decide whether to continue or return control to the planner. Eq. 2.

$$\{0, 1\} \ni c_t^n \leftarrow CTRL(h_t, a_t, I_t^n) \quad (2)$$

If $c_t^n = 1$, then the action is repeated and CTRL is applied to the next frame. Otherwise, if $c_t^n = 0$ or the maximum of 5 controller times is reached, control will be returned to the planner.

**Question Answering.** After the agent decides to stop or has taken the maximum number of actions (= 100), the execution question answering module provides the answer based on the sequence of frames $I_1, \ldots, I_T^n$ The agent observed. The reply module calculates the image problem similarity for each of the last 5 frames by (a) the dot product between the image features (aligned by the fc layer with the problem features) and (b) the problem code $Q$. These $I - Q$ similarities are converted to attentional weights by softmax and combine the attention-weighted image features with $Q$ (via concatenation) and through the softmax classifier to predict the distribution in 172 answers.

### 2.1. Imitation Learning and Reward Shaping

The Abhishek team uses a two-stage training process. First, the navigation and response modules are independently trained, using simulation/supervised learning to automatically generate navigation expert presentations. Second, the navigation architecture uses strategy gradients to fine-tune.

**Independent Pretraining via Imitation Learning.** Most of the questions that EmbassyQA may ask do not have a natural "correct" navigation to answer them.

**Target-aware Navigational Fine-tuning.** The navigation and answering modules generated by imitative learning work well independently, but are not suitable for working well with each other. Therefore, instead of forcing the respondent to provide the correct answer based on a noisy or absent point of view, they instead freeze it and fine-tune the navigator only. Abhishek team offer two types of rewards for the navigator: the accuracy of the answers and the formation of rewards at the end of the navigation Harada *et al.* [3], providing intermediate rewards for rewards that are closer to the goal. The Abhishek team uses REINFORCE proposed by Williams [5] for training and uses the average bonus as a benchmark. Just like in imitation learning, we follow the course of distance between spawning and target location.

## References


[1] J. Andreas, D. Klein, and S. Levine. Modular multitask reinforcement learning with policy sketches. In *ICML*, 2017. 2

[2] J. Johnson, B. Hariharan, L. van der Maaten, L. Fei-Fei, C. L. Zitnick, and R. Girshick. CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning. In *CVPR*, 2017. 1

[3] A. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, 1999. 3

[4] S. Song, F. Yu, A. Zeng, A. X. Chang, M. Savva, and T. A. Funkhouser. Semantic scene completion from a single depth image. In *CVPR*, 2017. 1

[5] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229-256, 1992. 3