Reinforcement Learning for Mapping Instructions to Actions with Reward Learning

Dipendra Misra  
Dept. of Computer Science and Cornell Tech  
Cornell University  
New York, NY 10044  
dkm@cs.cornell.edu

Yoav Artzi  
Dept. of Computer Science and Cornell Tech  
Cornell University  
New York, NY 10044  
yoav@cs.cornell.edu

Introduction

Natural language is an efficient medium for non-expert users to specify tasks for robotic agents. To effectively map natural language instructions to actions, a robotic agent must solve natural language, sensing, and planning problems. For example, consider the Blocks environment and instructions in Figure 1 (Bisk, Yuret, and Marcu 2016). The agent observes the environment as an RGB image using a camera sensor. Given the RGB input, the agent must recognize the blocks and their layout. To understand the instruction, the agent must identify the block to move (Toyota block) and the destination (just right of the SRI block). This requires solving semantic and grounding problems. For example, consider the topmost instruction in the figure. The agent needs to identify and ground the phrase \textit{SRI block} referring to the block to move. It must resolve and ground the phrase \textit{SRI block} as a reference position, which is then modified by the spatial meaning recovered from \textit{the same row as or first open space to the right of} to identify the goal position. Finally, the agent needs to generate actions, for example moving the Toyota block around obstructing blocks.

Previous work assumed a symbolic environment representation (Chen and Mooney 2011; Artzi and Zettlemoyer 2013; Artzi, Das, and Petrov 2014; Misra et al. 2015; Mei, Bansal, and Walter 2016), or combined separately trained models to solve the different problems (Matuszek, Fox, and Koscher 2010; Tellex et al. 2011). We recently proposed a single-model approach for mapping instructions and visual observations to actions (Misra, Langford, and Artzi 2017). Our approach does not require intermediate representations, planning procedures, or training different models. Training relies on a reinforcement learning method approximated in a contextual bandit setting (Langford and Zhang 2007). During learning, the reward function requires access to the world state to evaluate task progress and completion. While this can be achieved by instrumenting the training environment (Levine et al. 2016), this is not always practical.

We address this limitation by learning a distance-based reward function (Popov et al. 2017) using distance learning, an efficient method for learning a distance function (Weinberger, Blitzer, and Saul 2006). Learning relies on simple statistics over execution states. Once learned, the reward function is computed using agent observations and does not require access to the world state. In this abstract, we describe our ongoing work towards this goal. We evaluate in the simulated Blocks environment (Bisk, Marcu, and Wong 2016), and empirically demonstrate the effectiveness of the learned reward. Our approach shows limited degradation in performance in comparison to a reward that has access to the world state, while outperforming supervised learning and common reinforcement learning methods.

Problem Description

Task Let $\mathcal{X}$ be the set of all instructions, $\mathcal{S}$ the set of all world states, and $\mathcal{A}$ the set of all actions. An instruction $\tilde{x} \in \mathcal{X}$ is a sequence $\langle x_1, \ldots, x_n \rangle$, where each $x_i$ is a token. The agent executes instructions by generating a sequence of actions, and indicates execution completion with the special action $\text{STOP}$. Action execution modifies the world state following a transition function $T : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$. The execution $\tilde{e}$ of an instruction $\tilde{x}$ starting from $s_1$ is an $m$-length sequence $\langle (s_1, a_1), \ldots, (s_m, a_m) \rangle$, where $s_j \in \mathcal{S}$, $a_j \in \mathcal{A}$, $T(s_j, a_j) = s_{j+1}$ and $a_m = \text{STOP}$. In Blocks (Figure 1), a state specifies all block positions. For each action, the agent moves a block on the plane in one of four directions (north, south, east, or west). There are 20 blocks, and 81 possible actions at each step, including \text{STOP}. For example, the action \text{TOYOTA-SOUTH} moves the Toyota block one step south.
Figure 2: Illustration of the policy architecture showing the 10th step in the execution of Place the Toyota east of SRI starting from the state in Figure 1. The inputs are the instruction \( \bar{x} \), the current state image \( I_{10} \), previous state images \( (I_{9} \text{ and } I_{8}) \), and the previous action \( a_9 \). The text and images are embedded with a recurrent neural network LSTM (Hochreiter and Schmidhuber 1997) and a convolutional neural network CNN (LeCun et al. 1998). The action is selected with a multi-layer perceptron.

**Data** We train and evaluate on Blocks. The data set includes 16,767 natural language instructions with a vocabulary of 1,426. The mean instruction length is 15.27, and the mean number of actions 15.4. When compared to common data sets (MacMahon, Stankiewics, and Kuipers 2006; Matuszek et al. 2012; Misra et al. 2015), the instructions are longer, have a larger vocabulary, and require a larger number of actions (Misra, Langford, and Artzi 2017, Table 1).

**Single-Model Approach**

**Model** The agent observes the world state using a camera sensor. Given a world state \( s \), the agent observes an RGB image \( I \in \mathcal{I} \) generated by the function \( \text{Img}(s) \). We distinguish between the world state \( s \) and the agent context \( \bar{s} \), which includes the instruction, the image observation \( \text{Img}(s) \), images of previous states and the previous action. To map instructions to actions, the agent reasons about the agent context to generate a sequence of actions. At each step, the agent generates a single action. We model the agent with a neural network policy. At each step \( j \), the network takes as input the current agent context \( \bar{s}_j \), and predicts the next action to execute \( a_j \). Figure 2 illustrates the policy network. For full details see Misra, Langford, and Artzi (2017).

**Learning** We estimate the policy parameters using reinforcement learning in a contextual bandit setting. In a contextual bandit setting, maximizing the immediate reward suffices and provides stronger theoretical guarantees than unconstrained reinforcement learning (Agarwal et al. 2014).

**Reward with an Instrumented Environment**

In Misra, Langford, and Artzi (2017), we define a simple task-completion reward computed from the world state. To leverage demonstrations of the desired system behavior, we use reward shaping (Ng, Harada, and Russell 1999; Wiewiora, Cottrell, and Elkan 2003). Computing this reward requires instrumenting the training environment, a challenging engineering task in complex domains.

**Reward Learning with Metric Learning**

We adopt an inverse reinforcement learning (Ng and Russell 2000, IRL) approach and learn a reward function that is computed from the agent context. The original reward function is based on comparing world state and computing distances between them. Therefore, we cast the reward learning problem as learning a distant metric between world states, as observed in the agent context. Formally, our goal is to learn a distance function \( d_{\theta} : \mathcal{I} \times \mathcal{I} \to \mathbb{R} \) with parameters \( \theta \). We define \( d_\theta(I_1, I_2) = \| \phi_\theta(I_1) - \phi_\theta(I_2) \|_2 \), where \( \phi \) is a convolutional neural network with parameters \( \theta \). To learn the distance function, we assume access to a dataset of image triplets \( \{ (I_1^{(n)}, I_2^{(n)}, I_3^{(n)}) \}_{n=1}^N \), where the state of \( I_1^{(n)} \) is closer to state of \( I_2^{(n)} \) than the state of \( I_3^{(n)} \). We minimize the triplet loss (Weinberger, Blitzer, and Saul 2006):

\[
L = \frac{1}{N} \sum_{n=1}^N \max\{0, d_\theta(I_1^{(n)}, I_2^{(n)}) - d_\theta(I_1^{(n)}, I_3^{(n)}) + 1\}.
\]

To generate triplets we assume access to executions with progress meta data. The progress data includes for each execution step if it is closer to the final state than the previous step or further. A state may be further from the final state than the previous if the agent was moving further to, for example, go around an obstacle. We generate triplets for each pair of adjacent states and the final execution state. The image of the final state is \( I_a \), and the images of the next state and the current state are used as \( I_+ \) and \( I_- \), or vice versa, based on the progress meta data. The reward function is then defined as a potential difference \( R(I, a, I', I) = d_\theta(I_g, I) - d_\theta(I_g, I') \) where \( I, I' \) and \( I_g \) are the images of the current, next, and goal states.

**Results** Table 1 shows current development results. We measure execution error as the distance between the final and goal states, normalized by the block size. We report the mean error of following the demonstrations, random behavior (RANDOM), supervised learning (SUPERVISED), two reinforcement learning baselines (REINFORCE and DQN), our contextual bandit approach with instrumented training environment, and our approach with a learned reward.

**Conclusion**

We use metric learning to induce a reward function for learning to map instructions to actions. The reward function does not require the world state, and enables reinforcement learning without instrumenting the training environment. Our contextual bandit learning approach is designed for a few-samples regime. When the number of samples is unbounded, the drawbacks observed in this scenario for optimizing longer term reward do not hold.
References


