Mapping Instructions and Visual Observations to Actions with Reinforcement Learning

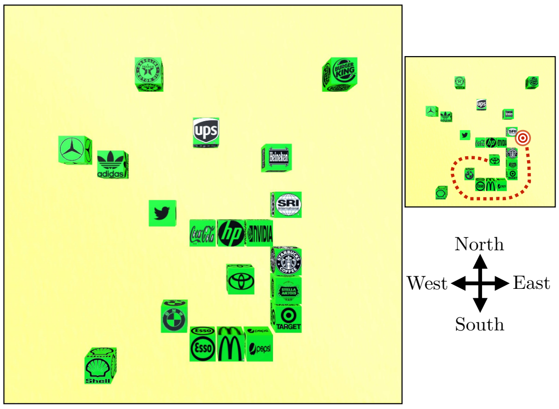
Dipendra Misra[†], **John Langford**[‡], **and Yoav Artzi**[†]

[†] Dept. of Computer Science and Cornell Tech, Cornell University, New York, NY 10044
{dkm, yoav}@cs.cornell.edu

[‡] Microsoft Research, New York, NY 10011
jcl@microsoft.com

Abstract

We propose to directly map raw visual observations and text input to actions for instruction execution. While existing approaches assume access to structured environment representations or use a pipeline of separately trained models, we learn a single model to jointly reason about linguistic and visual input. We use reinforcement learning in a contextual bandit setting to train a neural network agent. To guide the agent's exploration, we use reward shaping with different forms of supervision. Our approach does not require intermediate representations, planning procedures, or training different models. We evaluate in a simulated environment, and show significant improvements over supervised learning and common reinforcement learning variants.



Put the Toyota block in the same row as the SRI block, in the first open space to the right of the SRI block

Move Toyota to the immediate right of SRI, evenly aligned and slightly separated

Move the Toyota block around the pile and place it just to the right of the SRI block

Place Toyota block just to the right of The SRI Block

Toyota, right side of SRI

Figure 1: Instructions in the Blocks environment. The instructions all describe the same task. Given the observed RGB image of the start state (large image), our goal is to execute such instructions. In this task, the direct-line path to the target position is blocked, and the agent must plan and move the Toyota block around. The small image marks the target and an example path, which includes 34 steps.

1 Introduction

An agent executing natural language instructions requires robust understanding of language and its environment. Existing approaches addressing this problem assume structured environment representations (e.g.,. Chen and Mooney, 2011; Mei et al., 2016), or combine separately trained models (e.g., Matuszek et al., 2010; Tellex et al., 2011), including for language understanding and visual reasoning. We propose to directly map text and raw image input to actions with a single learned model. This approach offers multiple benefits, such as not requiring intermediate representations, planning procedures, or training multiple models.

Figure 1 illustrates the problem in the Blocks environment (Bisk et al., 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 the phrase referring to the block to move, *Toyota block*, and ground it. It must resolve and ground the phrase *SRI block* as a reference position, which is then modified by the spatial meaning recovered from *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.

To address these challenges with a single model, we design a neural network agent. The agent executes instructions by generating a sequence of actions. At each step, the agent takes as input the instruction text, observes the world as an RGB image, and selects the next action. Action execution changes the state of the world. Given an observation of the new world state, the agent selects the next action. This process continues until the agent indicates execution completion. When selecting actions, the agent jointly reasons about its observations and the instruction text. This enables decisions based on close interaction between observations and linguistic input.

We train the agent with different levels of supervision, including complete demonstrations of the desired behavior and annotations of the goal state only. While the learning problem can be easily cast as a supervised learning problem, learning only from the states observed in the training data results in poor generalization and failure to recover from test errors. We use reinforcement learning (Sutton and Barto, 1998) to observe a broader set of states through exploration. Following recent work in robotics (e.g., Levine et al., 2016; Rusu et al., 2016), we assume the training environment, in contrast to the test environment, is instrumented and provides access to the state. This enables a simple problem reward function that uses the state and provides positive reward on task completion only. This type of reward offers two important advantages: (a) it is a simple way to express the ideal agent behavior we wish to achieve, and (b) it creates a platform to add training data information.

We use reward shaping (Ng et al., 1999) to exploit the training data and add to the reward additional information. The modularity of shaping allows varying the amount of supervision, for example by using complete demonstrations for only a fraction of the training examples. Shaping also naturally associates actions with immediate reward. This enables learning in a contextual bandit setting (Auer et al., 2002; Langford and Zhang, 2007), where optimizing the immediate reward is sufficient and has better sample complexity than unconstrained reinforcement learning (Agarwal et al., 2014).

We evaluate with the block world environment and data of Bisk et al. (2016), where each instruction moves one block (Figure 1). While the original task focused on source and target prediction only, we build an interactive simulator and formulate the task of predicting the complete sequence of actions. At each step, the agent must select between 81 actions with 15.4 steps required to complete a task on average, significantly more than existing environments (e.g., Chen and Mooney, 2011). Our experiments demonstrate that our reinforcement learning approach effectively reduces execution error by 24% over standard supervised learning and 34-39% over common reinforcement learning techniques. Our simulator, code, models, and execution videos are available at: `https://github.com/clic-lab/blocks`.

2 Technical Overview

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 $\bar{x} \in \mathcal{X}$ 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 STOP. Action execution modifies the world state following a transition function $T : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$. The execution $\bar{e}$ of an instruction $\bar{x}$ 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 $a_m = \texttt{STOP}$. In Blocks (Figure 1), a state specifies the positions of all blocks. For each action, the agent moves a single 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 STOP. For example, to correctly execute the instructions in the figure, the agent's likely first action is TOYOTA-WEST, which moves the Toyota block one step west. Blocks can not move over or through other blocks.

Model The agent observes the world state via a visual sensor (i.e., a camera). Given a world state s, the agent observes an RGB image I generated by the function $\textsc{Img}(s)$. We distinguish between the world state s and the *agent context*[1] $\tilde{s}$, which includes the instruction, the observed image $\textsc{Img}(s)$, images of previous states, and the previous action. To map instructions to actions, the agent reasons about the agent context $\tilde{s}$ to generate a sequence of actions. At each step, the agent generates a single action. We model the agent with a

[1] We use the term *context* similar to how it is used in the contextual bandit literature to refer to the information available for decision making. While agent contexts capture information about the world state, they do not include physical information, except as captured by observed images.

neural network policy. At each step j, the network takes as input the current agent context $\tilde{s}_j$, and predicts the next action to execute a_j. We formally define the agent context and model in Section 4.

Learning We assume access to training data with N examples $\{(\bar{x}^{(i)}, s_1^{(i)}, \bar{e}^{(i)})\}_{i=1}^{N}$, where $\bar{x}^{(i)}$ is an instruction, $s_1^{(i)}$ is a start state, and $\bar{e}^{(i)}$ is an execution demonstration of $\bar{x}^{(i)}$ starting at $s_1^{(i)}$. We use policy gradient (Section 5) with reward shaping derived from the training data to increase learning speed and exploration effectiveness (Section 6). Following work in robotics (e.g., Levine et al., 2016), we assume an instrumented environment with access to the world state to compute the reward during training only. We define our approach in general terms with demonstrations, but also experiment with training using goal states.

Evaluation We evaluate task completion error on a test set $\{(\bar{x}^{(i)}, s_1^{(i)}, s_g^{(i)})\}_{i=1}^{M}$, where $\bar{x}^{(i)}$ is an instruction, $s_1^{(i)}$ is a start state, and $s_g^{(i)}$ is the goal state. We measure execution error as the distance between the final execution state and $s_g^{(i)}$.

3 Related Work

Learning to follow instructions was studied extensively with structured environment representations, including with semantic parsing (Chen and Mooney, 2011; Kim and Mooney, 2012, 2013; Artzi and Zettlemoyer, 2013; Artzi et al., 2014a,b; Misra et al., 2015, 2016), alignment models (Andreas and Klein, 2015), reinforcement learning (Branavan et al., 2009, 2010; Vogel and Jurafsky, 2010), and neural network models (Mei et al., 2016). In contrast, we study the problem of an agent that takes as input instructions and raw visual input. Instruction following with visual input was studied with pipeline approaches that use separately learned models for visual reasoning (Matuszek et al., 2010, 2012; Tellex et al., 2011; Paul et al., 2016). Rather than decomposing the problem, we adopt a single-model approach and learn from instructions paired with demonstrations or goal states. Our work is related to Sung et al. (2015). While they use sensory input to select and adjust a trajectory observed during training, we are not restricted to training sequences. Executing instructions in non-learning settings has also received significant attention (e.g., Winograd, 1972; Webber et al., 1995; MacMahon et al., 2006).

Our work is related to a growing interest in problems that combine language and vision, including visual question answering (e.g., Antol et al., 2015; Andreas et al., 2016b,a), caption generation (e.g., Chen et al., 2015, 2016; Xu et al., 2015), and visual reasoning (Johnson et al., 2016; Suhr et al., 2017). We address the prediction of the next action given a world image and an instruction.

Reinforcement learning with neural networks has been used for various NLP tasks, including text-based games (Narasimhan et al., 2015; He et al., 2016), information extraction (Narasimhan et al., 2016), co-reference resolution (Clark and Manning, 2016), and dialog (Li et al., 2016).

Neural network reinforcement learning techniques have been recently studied for behavior learning tasks, including playing games (Mnih et al., 2013, 2015, 2016; Silver et al., 2016) and solving memory puzzles (Oh et al., 2016). In contrast to this line of work, our data is limited. Observing new states in a computer game simply requires playing it. However, our agent also considers natural language instructions. As the set of instructions is limited to the training data, the set of agent contexts seen during learning is constrained. We address the data efficiency problem by learning in a contextual bandit setting, which is known to be more tractable (Agarwal et al., 2014), and using reward shaping to increase exploration effectiveness. Zhu et al. (2017) address generalization of reinforcement learning to new target goals in visual search by providing the agent an image of the goal state. We address a related problem. However, we provide natural language and the agent must learn to recognize the goal state.

Reinforcement learning is extensively used in robotics (Kober et al., 2013). Similar to recent work on learning neural network policies for robot control (Levine et al., 2016; Schulman et al., 2015; Rusu et al., 2016), we assume an instrumented training environment and use the state to compute rewards during learning. Our approach adds the ability to specify tasks using natural language.

4 Model

We model the agent policy π with a neural network. The agent observes the instruction and an RGB image of the world. Given a world state s, the image I is generated using the function $\textsc{Img}(s)$. The instruction execution is generated one step at a time. At each step j, the agent observes an image I_j of the current world state s_j and the instruction $\bar{x}$, predicts the action a_j, and executes it to transition to the next state s_{j+1}.

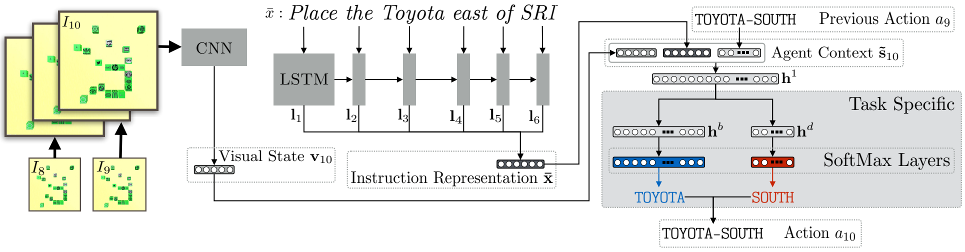

I_{10}
CNN
$\bar{x}$: *Place the Toyota east of SRI*
LSTM
$\mathbf{l}_1$
$\mathbf{l}_2$
$\mathbf{l}_3$
$\mathbf{l}_4$
$\mathbf{l}_5$
$\mathbf{l}_6$
I_8
I_9
Visual State $\mathbf{v}_{10}$
Instruction Representation $\bar{\mathbf{x}}$
TOYOTA-SOUTH Previous Action a_9
Agent Context $\tilde{\mathbf{s}}_{10}$
$\mathbf{h}^1$
Task Specific
$\mathbf{h}^b$
$\mathbf{h}^d$
SoftMax Layers
TOYOTA
SOUTH
TOYOTA-SOUTH Action a_{10}

Figure 2: Illustration of the policy architecture showing the 10th step in the execution of the instruction *Place the Toyota east of SRI* in the state from Figure 1. The network takes as input the instruction $\bar{x}$, image of the current state I_{10}, images of previous states I_8 and I_9 (with $K = 2$), and the previous action a_9. The text and images are embedded with LSTM and CNN. The actions are selected with the task specific multi-layer perceptron.

This process continues until STOP is predicted and the agent stops, indicating instruction completion. The agent also has access to K images of previous states and the previous action to distinguish between different stages of the execution (Mnih et al., 2015). Figure 2 illustrates our architecture.

Formally,[2] at step j, the agent considers an agent context $\tilde{s}_j$, which is a tuple $(\bar{x}, I_j, I_{j-1}, \ldots, I_{j-K}, a_{j-1})$, where $\bar{x}$ is the natural language instruction, I_j is an image of the current world state, the images $I_{j-1}, \ldots, I_{j-K}$ represent K previous states, and a_{j-1} is the previous action. The agent context includes information about the current state and the execution. Considering the previous action a_{j-1} allows the agent to avoid repeating failed actions, for example when trying to move in the direction of an obstacle. In Figure 2, the agent is given the instruction *Place the Toyota east of SRI*, is at the 10-th execution step, and considers $K = 2$ previous images.

We generate continuous vector representations for all inputs, and jointly reason about both text and image modalities to select the next action. We use a recurrent neural network (RNN; Elman, 1990) with a long short-term memory (LSTM; Hochreiter and Schmidhuber, 1997) recurrence to map the instruction $\bar{x} = \langle x_1, \ldots, x_n \rangle$ to a vector representation $\bar{\mathbf{x}}$. Each token x_i is mapped to a fixed dimensional vector with the learned embedding function $\psi(x_i)$. The instruction representation $\bar{\mathbf{x}}$ is computed by applying the LSTM recurrence to generate a sequence of hidden states $\mathbf{l}_i = \text{LSTM}(\psi(x_i), \mathbf{l}_{i-1})$, and computing the mean $\bar{\mathbf{x}} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{l}_i$ (Narasimhan et al., 2015). The current image I_j and previous images I_{j-1},…,I_{j-K} are concatenated along the channel dimension and embedded with a convolutional neural network (CNN) to generate the visual state $\mathbf{v}$ (Mnih et al., 2013). The last action a_{j-1} is embedded with the function $\psi_a(a_{j-1})$. The vectors $\mathbf{v}_j$, $\bar{\mathbf{x}}$, and $\psi_a(a_{j-1})$ are concatenated to create the agent context vector representation $\tilde{\mathbf{s}}_j = [\mathbf{v}_j, \bar{\mathbf{x}}, \psi_a(a_{j-1})]$.

To compute the action to execute, we use a feedforward perceptron that decomposes according to the domain actions. This computation selects the next action conditioned on the instruction text and observations from both the current world state and recent history. In the block world domain, where actions decompose to selecting the block to move and the direction, the network computes block and direction probabilities. Formally, we decompose an action a to direction a^D and block a^B. We compute the feedforward network:

$$\begin{aligned}
\mathbf{h}^1 &= \max(\mathbf{W}^{(1)}\tilde{\mathbf{s}}_j + \mathbf{b}^{(1)}, 0) \\
\mathbf{h}^D &= \mathbf{W}^{(D)}\mathbf{h}^1 + \mathbf{b}^{(D)} \\
\mathbf{h}^B &= \mathbf{W}^{(B)}\mathbf{h}^1 + \mathbf{b}^{(B)} \ ,
\end{aligned}$$

and the action probability is a product of the component probabilities:

$$\begin{aligned}
P(a_j^D = d \mid \bar{x}, s_j, a_{j-1}) &\propto \exp(\mathbf{h}_d^D) \\
P(a_j^B = b \mid \bar{x}, s_j, a_{j-1}) &\propto \exp(\mathbf{h}_b^B) \ .
\end{aligned}$$

At the beginning of execution, the first action a_0 is set to the special value NONE, and previous images are zero matrices. The embedding function ψ is a learned matrix. The function ψ_a concatenates the embeddings of a_{j-1}^D and a_{j-1}^B, which are obtained from learned matrices, to compute the embedding of a_{j-1}. The model parameters θ include $\mathbf{W}^{(1)}$, $\mathbf{b}^{(1)}$, $\mathbf{W}^{(D)}$, $\mathbf{b}^{(D)}$, $\mathbf{W}^{(B)}$, $\mathbf{b}^{(B)}$, the parameters of the LSTM recurrence, the parameters of the convolutional network CNN, and the embedding matrices. In our experiments (Section 7), all parameters are learned without external resources.

5 Learning

We use policy gradient for reinforcement learning (Williams, 1992) to estimate the parameters θ of the agent policy. We assume access to a

[2] We use bold-face capital letters for matrices and bold-face lowercase letters for vectors. Computed input and state representations use bold versions of the symbols. For example, $\bar{\mathbf{x}}$ is the computed representation of an instruction $\bar{x}$.

training set of N examples $\{(\bar{x}^{(i)}, s_1^{(i)}, \bar{e}^{(i)})\}_{i=1}^{N}$, where $\bar{x}^{(i)}$ is an instruction, $s_1^{(i)}$ is a start state, and $\bar{e}^{(i)}$ is an execution demonstration starting from $s_1^{(i)}$ of instruction $\bar{x}^{(i)}$. The main learning challenge is learning how to execute instructions given raw visual input from relatively limited data. We learn in a contextual bandit setting, which provides theoretical advantages over general reinforcement learning. In Section 8, we verify this empirically.

Reward Function The instruction execution problem defines a simple problem reward to measure task completion. The agent receives a positive reward when the task is completed, a negative reward for incorrect completion (i.e., `STOP` in the wrong state) and actions that fail to execute (e.g., when the direction is blocked), and a small penalty otherwise, which induces a preference for shorter trajectories. To compute the reward, we assume access to the world state. This learning setup is inspired by work in robotics, where it is achieved by instrumenting the training environment (Section 3). The agent, on the other hand, only uses the agent context (Section 4). When deployed, the system relies on visual observations and natural language instructions only. The reward function $R^{(i)} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is defined for each training example $(\bar{x}^{(i)}, s_1^{(i)}, \bar{e}^{(i)})$, $i = 1 \ldots N$:

$$R^{(i)}(s,a) = \begin{cases} 1.0 & \text{if } s = s_{m^{(i)}} \text{ and } a = \texttt{STOP} \\ -1.0 & s \neq s_{m^{(i)}} \text{ and } a = \texttt{STOP} \\ -1.0 & a \text{ fails to execute} \\ -\delta & \text{else} \end{cases} ,$$

where $m^{(i)}$ is the length of $\bar{e}^{(i)}$.

The reward function does not provide intermediate positive feedback to the agent for actions that bring it closer to its goal. When the agent explores randomly early during learning, it is unlikely to encounter the goal state due to the large number of steps required to execute tasks. As a result, the agent does not observe positive reward and fails to learn. In Section 6, we describe how reward shaping, a method to augment the reward with additional information, is used to take advantage of the training data and address this challenge.

Policy Gradient Objective We adapt the policy gradient objective defined by Sutton et al. (1999) to multiple starting states and reward functions:

$$\mathcal{J} = \frac{1}{N}\sum_{i=1}^{N} V_{\pi}^{(i)}(s_1^{(i)}) ,$$

where $V_{\pi}^{(i)}(s_1^{(i)})$ is the value given by $R^{(i)}$ starting from $s_1^{(i)}$ under the policy π. The summation expresses the goal of learning a behavior parameterized by natural language instructions.

Contextual Bandit Setting In contrast to most policy gradient approaches, we apply the objective to a contextual bandit setting where immediate reward is optimized rather than total expected reward. The primary theoretical advantage of contextual bandits is much tighter sample complexity bounds when comparing upper bounds for contextual bandits (Langford and Zhang, 2007) even with an adversarial sequence of contexts (Auer et al., 2002) to lower bounds (Krishnamurthy et al., 2016) or upper bounds (Kearns et al., 1999) for total reward maximization. This property is particularly suitable for the few-sample regime common in natural language problems. While reinforcement learning with neural network policies is known to require large amounts of training data (Mnih et al., 2015), the limited number of training sentences constrains the diversity and volume of agent contexts we can observe during training. Empirically, this translates to poor results when optimizing the total reward (REINFORCE baseline in Section 8). To derive the approximate gradient, we use the likelihood ratio method:

$$\nabla_{\theta}\mathcal{J} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}[\nabla_{\theta} \log \pi(\tilde{s}, a) R^{(i)}(s,a)] ,$$

where reward is computed from the world state but policy is learned on the agent context. We approximate the gradient using sampling.

This training regime, where immediate reward optimization is sufficient to optimize policy parameters θ, is enabled by the shaped reward we introduce in Section 6. While the objective is designed to work best with the shaped reward, the algorithm remains the same for any choice of reward definition including the original problem reward or several possibilities formed by reward shaping.

Entropy Penalty We observe that early in training, the agent is overwhelmed with negative reward and rarely completes the task. This results in the policy π rapidly converging towards a suboptimal deterministic policy with an entropy of 0. To delay premature convergence we add an entropy term to the objective (Williams and Peng, 1991; Mnih et al., 2016). The entropy term encourages a uniform distribution policy, and in practice stimulates exploration early during training. The regularized gradient is:

$$\nabla_{\theta}\mathcal{J} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}[\nabla_{\theta} \log \pi(\tilde{s}, a) R^{(i)}(s,a) + \lambda \nabla_{\theta} H(\pi(\tilde{s}, \cdot))] ,$$

Algorithm 1 Policy gradient learning

Input: Training set $\{(\bar{x}^{(i)}, s_1^{(i)}, \bar{e}^{(i)})\}_{i=1}^{N}$, learning rate μ, epochs T, horizon J, and entropy regularization term λ.

Definitions: $\textsc{Img}(s)$ is a camera sensor that reports an RGB image of state s. π is a probabilistic neural network policy parameterized by θ, as described in Section 4. $\textsc{Execute}(s, a)$ executes the action a at the state s, and returns the new state. $R^{(i)}$ is the reward function for example i. $\textsc{Adam}(\Delta)$ applies a per-feature learning rate to the gradient Δ (Kingma and Ba, 2014).

Output: Policy parameters θ.

```
 1: » Iterate over the training data.
 2: for $t = 1$ to $T$, $i = 1$ to $N$ do
 3:     $I_{1-K}, \ldots, I_0 = \vec{0}$
 4:     $a_0 = \texttt{NONE}$, $s_1 = s_1^{(i)}$
 5:     $j = 1$
 6:     » Rollout up to episode limit.
 7:     while $j \leq J$ and $a_j \neq \texttt{STOP}$ do
 8:         » Observe world and construct agent context.
 9:         $I_j = \textsc{Img}(s_j)$
10:         $\tilde{s}_j = (\bar{x}^{(i)}, I_j, I_{j-1}, \ldots, I_{j-K}, a_{j-1}^{d})$
11:         » Sample an action from the policy.
12:         $a_j \sim \pi(\tilde{s}_j, a)$
13:         $s_{j+1} = \textsc{Execute}(s_j, a_j)$
14:         » Compute the approximate gradient.
15:         $\Delta_j \leftarrow \nabla_\theta \log \pi(\tilde{s}_j, a_j) R^{(i)}(s_j, a_j)$
                      $+\lambda \nabla_\theta H(\pi(\tilde{s}_j, \cdot))$
16:         $j += 1$
17:     $\theta \leftarrow \theta + \mu \textsc{Adam}(\frac{1}{j} \sum_{j'=1}^{j} \Delta_{j'})$
18: return $\theta$
```

where $H(\pi(\tilde{s}, \cdot))$ is the entropy of π given the agent context $\tilde{s}$, λ is a hyperparameter that controls the strength of the regularization. While the entropy term delays premature convergence, it does not eliminate it. Similar issues are observed for vanilla policy gradient (Mnih et al., 2016).

Algorithm Algorithm 1 shows our learning algorithm. We iterate over the data T times. In each epoch, for each training example $(\bar{x}^{(i)}, s_1^{(i)}, \bar{e}^{(i)})$, $i = 1 \ldots N$, we perform a rollout using our policy to generate an execution (lines 7 - 16). The length of the rollout is bound by J, but may be shorter if the agent selected the STOP action. At each step j, the agent updates the agent context $\tilde{s}_j$ (lines 9 - 10), samples an action from the policy π (line 12), and executes it to generate the new world state s_{j+1} (line 13). The gradient is approximated using the sampled action with the computed reward $R^{(i)}(s_j, a_j)$ (line 15). Following each rollout, we update the parameters θ with the mean of the gradients using $\textsc{Adam}$ (Kingma and Ba, 2014).

6 Reward Shaping

Reward shaping is a method for transforming a reward function by adding a *shaping term* to the problem reward. The goal is to generate more informative updates by adding information to the reward. We use this method to leverage the training demonstrations, a common form of supervision for training systems that map language to actions. Reward shaping allows us to fully use this type of supervision in a reinforcement learning framework, and effectively combine learning from demonstrations and exploration.

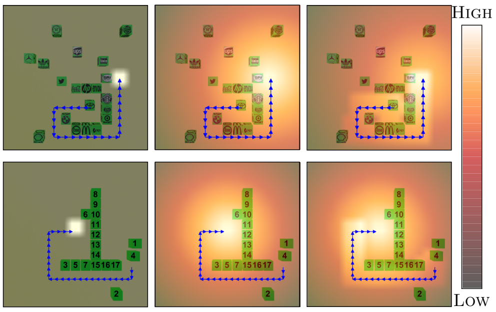


Figure 3: Visualization of the shaping potentials for two tasks. We show demonstrations (blue arrows), but omit instructions. To visualize the potentials intensity, we assume only the target block can be moved, while rewards and potentials are computed for any block movement. We illustrate the sparse problem reward (left column) as a potential function and consider only its positive component, which is focused on the goal. The middle column adds the distance-based potential. The right adds both potentials.

Adding an arbitrary shaping term can change the optimality of policies and modify the original problem, for example by making bad policies according to the problem reward optimal according to the shaped function.[3] Ng et al. (1999) and Wiewiora et al. (2003) outline potential-based terms that realize sufficient conditions for *safe* shaping.[4] Adding a shaping term is safe if the order of policies according to the shaped reward is identical to the order according to the original problem reward. While safe shaping only applies to optimizing the total reward, we show empirically the effectiveness of the safe shaping terms we design in a contextual bandit setting.

We introduce two shaping terms. The final shaped reward is a sum of them and the problem reward. Similar to the problem reward, we define example-specific shaping terms. We modify the reward function signature as required.

Distance-based Shaping (F_1) The first shaping term measures if the agent moved closer to the goal state. We design it to be a safe potential-based

[3] For example, adding a shaping term $F = -R$ will result in a shaped reward that is always 0, and any policy will be trivially optimal with respect to it.

[4] For convenience, we briefly overview the theorems of Ng et al. (1999) and Wiewiora et al. (2003) in Appendix A.

term (Ng et al., 1999):

$$F_1^{(i)}(s_j, a_j, s_{j+1}) = \phi_1^{(i)}(s_{j+1}) - \phi_1^{(i)}(s_j) \ .$$

The potential $\phi_1^{(i)}(s)$ is proportional to the negative distance from the goal state $s_g^{(i)}$. Formally, $\phi_1^{(i)}(s) = -\eta \| s - s_g^{(i)} \|$, where η is a constant scaling factor, and $\|.\|$ is a distance metric. In the block world, the distance between two states is the sum of the Euclidean distances between the positions of each block in the two states, and η is the inverse of block width. The middle column in Figure 3 visualizes the potential $\phi_1^{(i)}$.

Trajectory-based Shaping (F_2) Distance-based shaping may lead the agent to sub-optimal states, for example when an obstacle blocks the direct path to the goal state, and the agent must temporarily increase its distance from the goal to bypass it. We incorporate complete trajectories by using a simplification of the shaping term introduced by Brys et al. (2015). Unlike F_1, it requires access to the previous state and action. It is based on the look-back advice shaping term of Wiewiora et al. (2003), who introduced safe potential-based shaping that considers the previous state and action. The second term is:

$$F_2^{(i)}(s_{j-1}, a_{j-1}, s_j, a_j) = \phi_2^{(i)}(s_j, a_j) - \phi_2^{(i)}(s_{j-1}, a_{j-1}) \ .$$

Given $\bar{e}^{(i)} = \langle (s_1, a_1), \dots, (s_m, a_m) \rangle$, to compute the potential $\phi_2^{(i)}(s, a)$, we identify the closest state s_j in $\bar{e}^{(i)}$ to s. If $\eta \| s_j - s \| < 1$ and $a_j = a$, $\phi_2^{(i)}(s, a) = 1.0$, else $\phi_2^{(i)}(s, a) = -\delta_f$, where δ_f is a penalty parameter. We use the same distance computation and parameter η as in F_1. When the agent is in a state close to a demonstration state, this term encourages taking the action taken in the related demonstration state. The right column in Figure 3 visualizes the effect of the potential $\phi_2^{(i)}$.

7 Experimental Setup

Environment We use the environment of Bisk et al. (2016). The original task required predicting the source and target positions for a single block given an instruction. In contrast, we address the task of moving blocks on the plane to execute instructions given visual input. This requires generating the complete sequence of actions needed to complete the instruction. The environment contains up to 20 blocks marked with logos or digits. Each block can be moved in four directions. Including the `STOP` action, in each step, the agent selects between 81 actions. The set of actions is constant and is not limited to the blocks present. The transition function is deterministic. The size of each block step is 0.04 of the board size. The agent observes the board from above. We adopt a relatively challenging setup with a large action space. While a simpler setup, for example decomposing the problem to source and target prediction and using a planner, is likely to perform better, we aim to minimize task-specific assumptions and engineering of separate modules. However, to better understand the problem, we also report results for the decomposed task with a planner.

Data Bisk et al. (2016) collected a corpus of instructions paired with start and goal states. Figure 1 shows example instructions. The original data includes instructions for moving one block or multiple blocks. Single-block instructions are relatively similar to navigation instructions and referring expressions. While they present much of the complexity of natural language understanding and grounding, they rarely display the planning complexity of multi-block instructions, which are beyond the scope of this paper. Furthermore, the original data does not include demonstrations. While generating demonstrations for moving a single block is straightforward, disambiguating action ordering when multiple blocks are moved is challenging. Therefore, we focus on instructions where a single block changes its position between the start and goal states, and restrict demonstration generation to move the changed block. The remaining data, and the complexity it introduces, provide an important direction for future work.

To create demonstrations, we compute the shortest paths. While this process may introduce noise for instructions that specify specific trajectories (e.g., *move SRI two steps north and ...*) rather than only describing the goal state, analysis of the data shows this issue is limited. Out of 100 sampled instructions, 92 describe the goal state rather than the trajectory. A secondary source of noise is due to discretization of the state space. As a result, the agent often can not reach the exact target position. The demonstrations error illustrates this problem (Table 3). To provide task completion reward during learning, we relax the state comparison, and consider states to be equal if the sum of block distances is under the size of one block.

The corpus includes 11,871/1,719/3,177 instructions for training/development/testing. Table 1 shows corpus statistic compared to the commonly used SAIL navigation corpus (MacMahon

	SAIL	Blocks
Number of instructions	3,237	16,767
Mean instruction length	7.96	15.27
Vocabulary	563	1,426
Mean trajectory length	3.12	15.4

Table 1: Corpus statistics for the block environment we use and the SAIL navigation domain.

et al., 2006; Chen and Mooney, 2011). While the SAIL agent only observes its immediate surroundings, overall the blocks domain provides more complex instructions. Furthermore, the SAIL environment includes only 400 states, which is insufficient for generalization with vision input. We compare to other data sets in Appendix D.

Evaluation We evaluate task completion error as the sum of Euclidean distances for each block between its position at the end of the execution and in the gold goal state. We divide distances by block size to normalize for the image size. In contrast, Bisk et al. (2016) evaluate the selection of the source and target positions independently.

Systems We report performance of ablations, the upper bound of following the demonstrations (Demonstrations), and five baselines: (a) Stop: the agent immediately stops, (b) Random: the agent takes random actions, (c) Supervised: supervised learning with maximum-likelihood estimate using demonstration state-action pairs, (d) DQN: deep Q-learning with both shaping terms (Mnih et al., 2015), and (e) REINFORCE: policy gradient with cumulative episodic reward with both shaping terms (Sutton et al., 1999). Full system details are given in Appendix B.

Parameters and Initialization Full details are in Appendix C. We consider $K = 4$ previous images, and horizon length $J = 40$. We initialize our model with the Supervised model.

8 Results

Table 2 shows development results. We run each experiment three times and report the best result. The Random and Stop baselines illustrate the task complexity of the task. Our approach, including both shaping terms in a contextual bandit setting, significantly outperforms the other methods. Supervised learning demonstrates lower performance. A likely explanation is test-time execution errors leading to unfamiliar states with poor later performance (Kakade and Langford, 2002), a form of the covariate shift problem. The low performance of REINFORCE and DQN illustrates the challenge of general reinforcement learning with limited data due to relatively high sample complexity (Kearns et al., 1999; Krishnamurthy et al., 2016). We also report results using ensembles of the three models.

Algorithm	**Distance Error**		**Min. Distance**	
	Mean	**Med.**	**Mean**	**Med.**
Demonstrations	0.35	0.30	0.35	0.30
Baselines				
Stop	5.95	5.71	5.95	5.71
Random	15.3	15.70	5.92	5.70
Supervised	4.65	4.45	3.72	3.26
REINFORCE	5.57	5.29	4.50	4.25
DQN	6.04	5.78	5.63	5.49
Our Approach	3.60	3.09	2.72	2.21
w/o Sup. Init	3.78	3.13	2.79	2.21
w/o Prev. Action	3.95	3.44	3.20	2.56
w/o F_1	4.33	3.74	3.29	2.64
w/o F_2	3.74	3.11	3.13	2.49
w/ Distance Reward	8.36	7.82	5.91	5.70
Ensembles				
Supervised	4.64	4.27	3.69	3.22
REINFORCE	5.28	5.23	4.75	4.67
DQN	5.85	5.59	5.60	5.46
Our Approach	**3.59**	**3.03**	**2.63**	**2.15**

Table 2: Mean and median (Med.) development results.

Algorithm	**Distance Error**		**Min. Distance**	
	Mean	**Med.**	**Mean**	**Med.**
Demonstrations	0.37	0.31	0.37	0.31
Stop	6.23	6.12	6.23	6.12
Random	15.11	15.35	6.21	6.09
Ensembles				
Supervised	4.95	4.53	3.82	3.33
REINFORCE	5.69	5.57	5.11	4.99
DQN	6.15	5.97	5.86	5.77
Our Approach	**3.78**	**3.14**	**2.83**	**2.07**

Table 3: Mean and median (Med.) test results.

We ablate different parts of our approach. Ablations of supervised initialization (our approach w/o sup. init) or the previous action (our approach w/o prev. action) result in increase in error. While the contribution of initialization is modest, it provides faster learning. On average, after two epochs, we observe an error of 3.94 with initialization and 6.01 without. We hypothesize that the F_2 shaping term, which uses full demonstrations, helps to narrow the gap at the end of learning. Without supervised initialization and F_2, the error increases to 5.45 (the 0% point in Figure 4). We observe the contribution of each shaping term and their combination. To study the benefit of potential-based shaping, we experiment with a negative distance-to-goal reward. This reward replaces the problem reward and encourages getting closer to the goal (our approach w/distance reward). With this reward, learning fails to converge, leading to a relatively high error.

Figure 4 shows our approach with varying

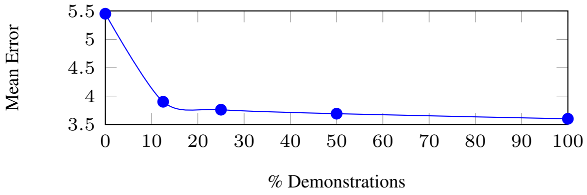


Figure 4: Mean distance error as a function of the ratio of training examples that include complete trajectories. The rest of the data includes the goal state only.

amount of supervision. We remove demonstrations from both supervised initialization and the F_2 shaping term. For example, when only 25% are available, only 25% of the data is available for initialization and the F_2 term is only present for this part of the data. While some demonstrations are necessary for effective learning, we get most of the benefit with only 12.5%.

Table 3 provides test results, using the ensembles to decrease the risk of overfitting the development. We observe similar trends to development result with our approach outperforming all baselines. The remaining gap to the demonstrations upper bound illustrates the need for future work.

To understand performance better, we measure *minimal distance* (min. distance in Tables 2 and 3), the closest the agent got to the goal. We observe a strong trend: the agent often gets close to the goal and fails to stop. This behavior is also reflected in the number of steps the agent takes. While the mean number of steps in development demonstrations is 15.2, the agent generates on average 28.7 steps, and 55.2% of the time it takes the maximum number of allowed steps (40). Testing on the training data shows an average 21.75 steps and exhausts the number of steps 29.3% of the time. The mean number of steps in training demonstrations is 15.5. This illustrates the challenge of learning how to be behave at an absorbing state, which is observed relatively rarely during training. This behavior also shows in our video.[5]

We also evaluate a supervised learning variant that assumes a perfect planner.[6] This setup is similar to Bisk et al. (2016), except using raw image input. It allows us to roughly understand how well the agent generates actions. We observe a mean error of 2.78 on the development set, an improvement of almost two points over supervised learning with our approach. This illustrates the complexity of the complete problem.

We conduct a shallow linguistic analysis to understand the agent behavior with regard to differences in the language input. As expected, the agent is sensitive to unknown words. For instructions without unknown words, the mean development error is 3.49. It increases to 3.97 for instructions with a single unknown word, and to 4.19 for two.[7] We also study the agent behavior when observing new phrases composed of known words by looking at instructions with new n-grams and no unknown words. We observe no significant correlation between performance and new bi-grams and tri-grams. We also see no meaningful correlation between instruction length and performance. Although counterintuitive given the linguistic complexities of longer instructions, it aligns with results in machine translation (Luong et al., 2015).

9 Conclusions

We study the problem of learning to execute instructions in a situated environment given only raw visual observations. Supervised approaches do not explore adequately to handle test time errors, and reinforcement learning approaches require a large number of samples for good convergence. Our solution provides an effective combination of both approaches: reward shaping to create relatively stable optimization in a contextual bandit setting, which takes advantage of a signal similar to supervised learning, with a reinforcement basis that admits substantial exploration and easy avenues for smart initialization. This combination is designed for a few-samples regime, as we address. When the number of samples is unbounded, the drawbacks observed in this scenario for optimizing longer term reward do not hold.

Acknowledgments

This research was supported by a Google Faculty Award, an Amazon Web Services Research Grant, and a Schmidt Sciences Research Award. We thank Alane Suhr, Luke Zettlemoyer, and the anonymous reviewers for their helpful feedback, and Claudia Yan for technical help. We also thank the Cornell NLP group and the Microsoft Research Machine Learning NYC group for their support and insightful comments.

[5] https://github.com/clic-lab/blocks

[6] As there is no sequence of decisions, our reinforcement approach is not appropriate for the planner experiment. The architecture details are described in Appendix B.

[7] This trend continues, although the number of instructions is too low (< 20) to be reliable.

References

Alekh Agarwal, Daniel J. Hsu, Satyen Kale, John Langford, Lihong Li, and Robert E. Schapire. 2014. Taming the monster: A fast and simple algorithm for contextual bandits. In *Proceedings of the International Conference on Machine Learning*.

Jacob Andreas and Dan Klein. 2015. Alignment-based compositional semantics for instruction following. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. https://doi.org/10.18653/v1/D15-1138.

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. 2016a. Learning to compose neural networks for question answering. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. https://doi.org/10.18653/v1/N16-1181.

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. 2016b. Neural module networks. In *Conference on Computer Vision and Pattern Recognition*.

Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C. Lawrence Zitnick, and Devi Parikh. 2015. VQA: Visual question answering. In *International Journal of Computer Vision*.

Yoav Artzi, Dipanjan Das, and Slav Petrov. 2014a. Learning compact lexicons for CCG semantic parsing. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*. https://doi.org/10.3115/v1/D14-1134.

Yoav Artzi, Maxwell Forbes, Kenton Lee, and Maya Cakmak. 2014b. Programming by demonstration with situated semantic parsing. In *AAAI Fall Symposium Series*.

Yoav Artzi and Luke Zettlemoyer. 2013. Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association of Computational Linguistics* 1:49–62. http://aclweb.org/anthology/Q13-1005.

Peter Auer, Nicolò Cesa-Bianchi, Yoav Freund, and Robert E. Schapire. 2002. The nonstochastic multiarmed bandit problem. *SIAM J. Comput.* 32(1):48–77.

Yonatan Bisk, Deniz Yuret, and Daniel Marcu. 2016. Natural language communication with robots. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. https://doi.org/10.18653/v1/N16-1089.

S.R.K. Branavan, Harr Chen, Luke Zettlemoyer, and Regina Barzilay. 2009. Reinforcement learning for mapping instructions to actions. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*. http://aclweb.org/anthology/P09-1010.

S.R.K. Branavan, Luke Zettlemoyer, and Regina Barzilay. 2010. Reading between the lines: Learning to map high-level instructions to commands. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*. http://aclweb.org/anthology/P10-1129.

Tim Brys, Anna Harutyunyan, Halit Bener Suay, Sonia Chernova, Matthew E. Taylor, and Ann Nowé. 2015. Reinforcement learning from demonstration through shaping. In *Proceedings of the International Joint Conference on Artificial Intelligence*.

David L. Chen and Raymond J. Mooney. 2011. Learning to interpret natural language navigation instructions from observations. In *Proceedings of the National Conference on Artificial Intelligence*.

Wenhu Chen, Aurélien Lucchi, and Thomas Hofmann. 2016. Bootstrap, review, decode: Using out-of-domain textual data to improve image captioning. *CoRR* abs/1611.05321.

Xinlei Chen, Hao Fang, Tsung-Yi Lin, Ramakrishna Vedantam, Saurabh Gupta, Piotr Dollár, and C. Lawrence Zitnick. 2015. Microsoft COCO captions: Data collection and evaluation server. *CoRR* abs/1504.00325.

Kevin Clark and D. Christopher Manning. 2016. Deep reinforcement learning for mention-ranking coreference models. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. http://aclweb.org/anthology/D16-1245.

Jeffrey L. Elman. 1990. Finding structure in time. *Cognitive Science* 14:179–211.

Ji He, Jianshu Chen, Xiaodong He, Jianfeng Gao, Lihong Li, Li Deng, and Mari Ostendorf. 2016. Deep reinforcement learning with a natural language action space. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. https://doi.org/10.18653/v1/P16-1153.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9.

Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C. Lawrence Zitnick, and Ross B. Girshick. 2016. CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning. *CoRR* abs/1612.06890.

Sham Kakade and John Langford. 2002. Approximately optimal approximate reinforcement learning. In *Machine Learning, Proceedings of the Nineteenth International Conference (ICML 2002), University of New South Wales, Sydney, Australia, July 8-12, 2002*.

Michael Kearns, Yishay Mansour, and Andrew Y. Ng. 1999. A sparse sampling algorithm for near-optimal planning in large markov decision processes. In *Proeceediings of the International Joint Conference on Artificial Intelligence*.

Joohyun Kim and Raymond Mooney. 2012. Unsupervised PCFG induction for grounded language learning with highly ambiguous supervision. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. http://aclweb.org/anthology/D12-1040.

Joohyun Kim and Raymond Mooney. 2013. Adapting discriminative reranking to grounded language learning. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. http://aclweb.org/anthology/P13-1022.

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations*.

Jens Kober, J. Andrew Bagnell, and Jan Peters. 2013. Reinforcement learning in robotics: A survey. *International Journal of Robotics Research* 32:1238–1274.

Akshay Krishnamurthy, Alekh Agarwal, and John Langford. 2016. PAC reinforcement learning with rich observations. In *Advances in Neural Information Processing Systems*.

John Langford and Tong Zhang. 2007. The epoch-greedy algorithm for multi-armed bandits with side information. In *Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 3-6, 2007*.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. 2016. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research* 17.

Jiwei Li, Will Monroe, Alan Ritter, Dan Jurafsky, Michel Galley, and Jianfeng Gao. 2016. Deep reinforcement learning for dialogue generation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. http://aclweb.org/anthology/D16-1127.

Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. http://aclweb.org/anthology/D15-1166.

Matthew MacMahon, Brian Stankiewics, and Benjamin Kuipers. 2006. Walk the talk: Connecting language, knowledge, action in route instructions. In *Proceedings of the National Conference on Artificial Intelligence*.

Cynthia Matuszek, Dieter Fox, and Karl Koscher. 2010. Following directions using statistical machine translation. In *Proceedings of the international conference on Human-robot interaction*.

Cynthia Matuszek, Evan Herbst, Luke S. Zettlemoyer, and Dieter Fox. 2012. Learning to parse natural language commands to a robot control system. In *Proceedings of the International Symposium on Experimental Robotics*.

Hongyuan Mei, Mohit Bansal, and R. Matthew Walter. 2016. What to talk about and how? selective generation using lstms with coarse-to-fine alignment. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. https://doi.org/10.18653/v1/N16-1086.

Dipendra K. Misra, Jaeyong Sung, Kevin Lee, and Ashutosh Saxena. 2016. Tell me dave: Context-sensitive grounding of natural language to manipulation instructions. *The International Journal of Robotics Research* 35(1-3):281–300.

Kumar Dipendra Misra, Kejia Tao, Percy Liang, and Ashutosh Saxena. 2015. Environment-driven lexicon induction for high-level instructions. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. https://doi.org/10.3115/v1/P15-1096.

Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *Proceedings of the International Conference on Machine Learning*.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. 2013. Playing atari with deep reinforcement learning. In *Advances in Neural Information Processing Systems*.

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

Karthik Narasimhan, Tejas Kulkarni, and Regina Barzilay. 2015. Language understanding for text-based games using deep reinforcement learning. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. https://doi.org/10.18653/v1/D15-1001.

Karthik Narasimhan, Adam Yala, and Regina Barzilay. 2016. Improving information extraction by acquiring external evidence with reinforcement learning. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. http://aclweb.org/anthology/D16-1261.

Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. 1999. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the International Conference on Machine Learning*.

Junhyuk Oh, Valliappa Chockalingam, Satinder P. Singh, and Honglak Lee. 2016. Control of memory, active perception, and action in minecraft. In *Proceedings of the International Conference on Machine Learning*.

Rohan Paul, Jacob Arkin, Nicholas Roy, and Thomas M. Howard. 2016. Efficient grounding of abstract spatial concepts for natural language interaction with robot manipulators. In *Robotics: Science and Systems*.

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

John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. 2015. Trust region policy optimization .

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. 2016. Mastering the game of go with deep neural networks and tree search. *Nature* 529 7587:484–9.

Alane Suhr, Mike Lewis, James Yeh, and Yoav Artzi. 2017. A corpus of compositional language for visual reasoning. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics.

Jaeyong Sung, Seok Hyun Jin, and Ashutosh Saxena. 2015. Robobarista: Object part based transfer of manipulation trajectories from crowd-sourcing in 3d pointclouds. In *International Symposium on Robotics Research*.

Richard S. Sutton and Andrew G. Barto. 1998. Reinforcement learning: An introduction. *IEEE Trans. Neural Networks* 9:1054–1054.

Richard S. Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. 1999. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*.

Stefanie Tellex, Thomas Kollar, Steven Dickerson, Matthew Walter, Ashis G. Banerjee, Seth Teller, and Nicholas Roy. 2011. Understanding natural language commands for robotic navigation and mobile manipulation. In *Proceedings of the National Conference on Artificial Intelligence*.

Adam Vogel and Daniel Jurafsky. 2010. Learning to follow navigational directions. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*. http://aclweb.org/anthology/P10-1083.

Bonnie Webber, Norman Badler, Barbara Di Eugenio, Christopher Geib, Libby Levison, and Michael Moore. 1995. Instructions, intentions and expectations. *Artificial Intelligence* 73(1):253–269.

Eric Wiewiora, Garrison W. Cottrell, and Charles Elkan. 2003. Principled methods for advising reinforcement learning agents. In *Proceedings of the International Conference on Machine Learning*.

Ronald J. Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning* 8.

Ronald J Williams and Jing Peng. 1991. Function optimization using connectionist reinforcement learning algorithms. *Connection Science* 3(3):241–268.

Terry Winograd. 1972. Understanding natural language. *Cognitive Psychology* 3(1):1–191.

Kelvin Xu, Jimmy Ba, Jamie Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. 2015. Show, attend and tell: Neural image caption generation with visual attention. In *Proceedings of the International Conference on Machine Learning*.

Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J. Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. 2017. Target-driven visual navigation in indoor scenes using deep reinforcement learning.

A Reward Shaping Theorems

In Section 6, we introduce two reward shaping terms. We follow the safe-shaping theorems of Ng et al. (1999) and Wiewiora et al. (2003). The theorems outline potential-based terms that realize sufficient conditions for *safe* shaping. Applying safe terms guarantees the order of policies according to the original problem reward does not change. While the theory only applies when optimizing the total reward, we show empirically the effectiveness of the safe shaping terms in a contextual bandit setting. For convenience, we provide the definitions of potential-based shaping terms and the theorems introduced by Ng et al. (1999) and Wiewiora et al. (2003) using our notation. We refer the reader to the original papers for the full details and proofs.

The distance-based shaping term F_1 is defined based on the theorem of Ng et al. (1999):

Definition. *A shaping term $F : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ is potential-based if there exists a function $\phi : \mathcal{S} \rightarrow \mathbb{R}$ such that, at time j, $F(s_j, a_j, s_{j+1}) = \gamma\phi(s_{j+1}) - \phi(s_j)$, $\forall s_j, s_{j+1} \in \mathcal{S}$ and $a_j \in \mathcal{A}$, where $\gamma \in [0, 1]$ is a future reward discounting factor. The function ϕ is the potential function of the shaping term F.*

Theorem. *Given a reward function $R(s_j, a_j)$, if the shaping term is potential-based, the shaped reward $R_F(s_j, a_j, s_{j+1}) = R(s_j, a_j) + F(s_j, a_j, s_{j+1})$ does not modify the total order of policies.*

In the definition of F_1, we set the discounting term γ to 1.0 and omit it.

The trajectory-based shaping term F_2 follows the shaping term introduced by Brys et al. (2015). To define it, we use the look-back advice shaping term of Wiewiora et al. (2003), who extended the potential-based term of Ng et al. (1999) for terms that consider the previous state and action:

Definition. *A shaping term $F : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is potential-based if there exists a function $\phi : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ such that, at time j, $F(s_{j-1}, a_{j-1}, s_j, a_j) = \gamma\phi(s_j, a_j) - \phi(s_{j-1}, a_{j-1})$, $\forall s_j, s_{j-1} \in \mathcal{S}$ and $a_j, a_{j-1} \in \mathcal{A}$, where $\gamma \in [0, 1]$ is a future reward discounting factor. The function ϕ is the potential function of the shaping term F.*

Theorem. *Given a reward function $R(s_j, a_j)$, if the shaping term is potential-based, the shaped reward $R_F(s_{j-1}, a_{j-1}, s_j, a_j) = R(s_j, a_j) + F(s_{j-1}, a_{j-1}, s_j, a_j)$ does not modify the total order of policies.*

In the definition of F_2 as well, we set the discounting term γ to 1.0 and omit it.

B Evaluation Systems

We implement multiple systems for evaluation.

STOP The agent performs the `STOP` action immediately at the beginning of execution.

RANDOM The agent samples actions uniformly until `STOP` is sampled or J actions were sampled, where J is the execution horizon.

SUPERVISED Given the training data with N instruction-state-execution triplets, we generate training data of instruction-state-action triplets and optimize the log-likelihood of the data. Formally, we optimize the objective:

$$J = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{m^{(i)}} \log \pi(\tilde{s}_j^{(i)}, a_j^{(i)}) \ ,$$

where $m^{(i)}$ is the length of the execution $\bar{e}^{(i)}$, $\tilde{s}_j^{(i)}$ is the agent context at step j in sample i, and $a_j^{(i)}$ is the demonstration action of step j in demonstration execution $\bar{e}^{(i)}$. Agent contexts are generated with the annotated previous actions (i.e., to generate previous images and the previous action). We use minibatch gradient descent with ADAM updates (Kingma and Ba, 2014).

DQN We use deep Q-learning (Mnih et al., 2015) to train a Q-network. We use the architecture described in Section 4, except replacing the task specific part with a single 81-dimension layer. In contrast to our probabilistic model, we do not decompose block and direction selection. We use the shaped reward function, including both F_1 and F_2. We use a replay memory of size 2,000 and an ϵ-greedy behavior policy to generate rollouts. We attenuate the value of ϵ from 1 to 0.1 in 100,000 steps and use prioritized sweeping for sampling. We also use a target network that is synchronized after every epoch.

REINFORCE We use the REINFORCE algorithm (Sutton et al., 1999) to train our agent. REINFORCE performs policy gradient learning with total reward accumulated over the roll-out as opposed to using immediate rewards as in our main approach. REINFORCE samples the total reward using monte-carlo sampling by performing a roll-out. We use the shaped reward function, including both F_1 and F_2 terms. Similar to our approach, we initialize with a SUPERVISED model and regularize the objective with the entropy of the policy. We do not use a reward baseline.

SUPERVISED with Oracle Planner We use a variant of our model assuming a perfect planner. The model predicts the block to move and its target position as a pair of coordinates. We modify the architecture in Section 4 to predict the block to move and its target position as a pair of coordinates. This model assumes that the sequence of actions is inferred from the predicted target position using an oracle planner. We train using supervised learning by maximizing the likelihood of the block being moved and minimizing the squared distance between the predicted target position and the annotated target position.

C Parameters and Initialization

C.1 Architecture Parameters

We use an RGB image of 120x120 pixels, and a convolutional neural network (CNN) with 4 layers. The first two layers apply 32 8×8 filters with a stride of 4, the third applies 32 4×4 filters with a stride of 2. The last layer performs an affine transformation to create a 200-dimension vector. We linearly scale all images to have zero mean and unit norm. We use a single layer RNN with 150-dimensional word embeddings and 250 LSTM units. The dimension of the action embedding ψ_a is 56, including 32 for embedding the block and 24 for embedding the directions. $\mathbf{W}^{(1)}$ is a 506×120 matrix and $\mathbf{b}^{(1)}$ is a 120-dimension vector. $\mathbf{W}^{(D)}$ is 120×20 for 20 blocks, and $\mathbf{W}^{(B)}$ is 120×5 for the four directions (north, south, east, west) and the STOP action. We consider $K = 4$ previous images, and use horizon length $J = 40$.

C.2 Initialization

Embedding matrices are initialized with a zero-mean unit-variance Gaussian distribution. All biases are initialized to $\mathbf{0}$. We use a zero-mean truncated normal distribution to initialize the CNN filters (0.005 variance) and CNN weights matrices (0.004 variance). All other weight matrices are initialized with a normal distribution (mean=0.0, standard deviation=0.01). The matrices used in the word embedding function ψ are initialized with a zero-mean normal distribution with standard deviation of 1.0. Action embedding matrices, which are used for ψ_a, are initialized with a zero-mean normal distribution with 0.001 standard deviation. We initialize policy gradient learning, including our approach, with parameters estimated using supervised learning for two epochs, except the direction parameters $\mathbf{W}^{(D)}$ and $\mathbf{b}^{(D)}$, which we learn from scratch. We found this initialization method to provide a good balance between strong initialization and not biasing the learning too much, which can result in limited exploration.

C.3 Learning Parameters

We use the distance error on a small validation set as stopping criteria. After each epoch, we save the model, and select the final model based on development set performance. While this method overfits the development set, we found it more reliable then using the small validation set alone. Our relatively modest performance degradation on the held-out set illustrates that our models generalize well. We set the reward and shaping penalties $\delta = \delta_f = 0.02$. The entropy regularization coefficient is $\lambda = 0.1$. The learning rate is $\mu = 0.001$ for supervised learning and $\mu = 0.00025$ for policy gradient. We clip the gradient at a norm of 5.0. All learning algorithms use a mini-batch of size 32 during training.

D Dataset Comparisons

We briefly review instruction following datasets in Table 4, including: Blocks (Bisk et al., 2016), SAIL (MacMahon et al., 2006; Chen and Mooney, 2011), Matuszek (Matuszek et al., 2012), and Misra (Misra et al., 2015). Overall, Blocks provides the largest training set and a relatively complex environment with well over 2.43^{18} possible states.[8] The most similar dataset is SAIL, which provides only partial observability of the environment (i.e., the agent observes what is around it only). However, SAIL is less complex on other dimensions related to the instructions, trajectories, and action space. In addition, while Blocks has a large number of possible states, SAIL includes only 400 states. The small number of states makes it difficult to learn vision models that generalize well. Misra (Misra et al., 2015) provides a parameterized action space (e.g., $\mathrm{grasp}(\mathrm{cup})$), which leads to a large number of potential actions. However, the corpus is relatively small.

E Common Questions

This is a list of potential questions following various decisions that we made. While we ablated and discussed all the crucial decisions in the paper, we decided to include this appendix to provide as much information as possible.

Is it possible to manually engineer a competitive reward function without shaping? Shaping is a principled approach to add information to a problem reward with relatively intuitive potential functions. Our experiments demonstrate its effectiveness. Investing engineering effort in designing

[8] We compute this loose lower bound on the number of states in the block world as $20! = 2.43^{18}$ (the number of block permutations). This is a very loose lower bound.

Name	# Samples	Vocabulary Size	Mean Instruction Length	# Actions	Mean Trajectory Length	Partially Observed
Blocks	16,767	1,426	15.27	81	15.4	No
SAIL	3,237	563	7.96	3	3.12	Yes
Matuszek	217	39	6.65	3	N/A	No
Misra	469	775	48.7	> 100	21.5	No

Table 4: Comparison of several related natural language instructions corpora.

a reward function specifically designed to the task is a potential alternative approach.

Are you using beam search? Why not? While using beam search can probably increase our performance, we chose to avoid it. We are motivated by robotic scenarios, where implementing beam search is a challenging task and often not possible. We distinguish between beam search and backtracking. Beam search is also incompatible with common assumptions of reinforcement learning, although it is often used during test with reinforcement learning systems.

Why are you using the mean of the LSTM hidden states instead of just the final state? We empirically tested both options. Using the mean worked better. This was also observed by Narasimhan et al. (2015). Understanding in which scenarios one technique is better than the other is an important question for future work.

Can you provide more details about initialization? Please see Appendix C.

Does the agent in the block world learn to move obstacles and other blocks? While the agent can move any block at any step, in practice, it rarely happens. The agent prefers to move blocks around obstacles rather than moving other blocks and moving them back into place afterwards. This behavior is learned from the data and shows even when we use only very limited amount of demonstrations. We hypothesize that in other tasks the agent is likely to learn that moving obstacles is advantageous, for example when demonstrations include moving obstacles.

Does the agent explicitly mark where it is in the instruction? We estimate that over 90% of the instructions describe the target position. Therefore, it is often not clear how much of the instruction was completed during the execution. The agent does not have an explicit mechanism to mark portions of the instruction that are complete. We briefly experimented with attention, but found that empirically it does not help in our domain. Designing an architecture to allows such considerations is an important direction for future work.

Does the agent know which blocks are present? Not all blocks are included in each task. The agent must infer which blocks are present from the image and instruction. The set of possible actions, which includes moving all possible blocks, does not change between tasks. If the agent chooses to move a block that is not present, the world state does not change.

Did you experiment with executing sequences of instruction? The Bisk et al. (2016) includes such instructions, right? The majority of existing corpora, including SAIL (Chen and Mooney, 2011; Artzi and Zettlemoyer, 2013; Mei et al., 2016), provide segmented sequences of instructions. Existing approaches take advantage of this segmentation during training. For example, Chen and Mooney (2011), Artzi and Zettlemoyer (2013), and Mei et al. (2016) all train on segmented data and test on sequences of instructions by doing inference on one sentence at a time. We are also able to do this. Similar to these approaches, we will likely suffer from cascading errors. The multi-instruction paragraphs in the Bisk et al. (2016) data are an open problem and present new challenges beyond just instruction length. For example, they often merge multiple block placements in one instruction (e.g, *put the SRI, HP, and Dell blocks in a row*). Since the original corpus does not provide trajectories and our automatic generation procedure is not able to resolve which block to move first, we do not have demonstrations for this data. The instructions also present a significantly more complex task. This is an important direction for future work, which illustrates the complexity and potential of the domain.

Potential-based shaping was proven to be safe when maximizing the total expected reward. Does this apply for the contextual bandit setting, where you maximize the immediate reward? The safe shaping theorems (Appendix A) do not hold in our contextual bandit setting. We show empirically that shaping works in practice. However, how and if it changes the order of policies is an open question.

How long does it take to train? How many frames the agent observes? The agent observes about 2.5 million frames. It takes 16 hours using 50% capacity of an Nvidia Pascal Titan X GPU to train using our approach. DQN takes more than twice the time for the same number of epochs. Supervised learning takes about 9 hours to converge. We also trained DQN for around four days, but did not observe improvement.

Did you consider initializing DQN with supervised learning? Initializing DQN with the probabilistic supervised model is challenging. Since DQN is not probabilistic it is not clear what this initialization means. Smart initialization of DQN is an important problem for future work.