**ABSTRACT**

Behavioral skills or policies for autonomous agents are conventionally learned from reward functions, via reinforcement learning, or from demonstrations, via imitation learning. However, both modes of task specification have their disadvantages: reward functions require manual engineering, while demonstrations require a human expert to be able to actually perform the task in order to generate the demonstration. Instruction following from natural language instructions provides an appealing alternative: in the same way that we can specify goals to other humans simply by speaking or writing, we would like to be able to specify tasks for our machines. However, a single instruction may be insufficient to fully communicate our intent or, even if it is, may be insufficient for an autonomous agent to actually understand how to perform the desired task. In this work, we propose an interactive formulation of the task specification problem, where iterative language corrections are provided to an autonomous agent, guiding it in acquiring the desired skill. Our proposed language-guided policy learning algorithm can integrate an instruction and a sequence of corrections to acquire new skills very quickly. In our experiments, we show that this method can enable a policy to follow instructions and corrections for simulated navigation and manipulation tasks, substantially outperforming direct, non-interactive instruction following.

1 **INTRODUCTION**

Behavioral skills or policies for autonomous agents are typically specified in terms of reward functions (in the case of reinforcement learning) or demonstrations (in the case of imitation learning). However, both reward functions and demonstrations have downsides as mechanisms for communicating goals. Reward functions must be engineered manually, which can be challenging in real-world environments, especially when the learned policies operate directly on raw sensory perception; sometimes, simply defining the goal of the task requires engineering the very perception system that end-to-end deep learning is supposed to acquire. Demonstrations sidestep this challenge, but require a human demonstrator to actually be able to perform the task, which can be cumbersome or even impossible. When humans must communicate goals to each other, we use language. Considerable research has also focused on building autonomous agents that can follow instructions provided via language (Janner et al. (2018); Andreas & Klein (2015); Fried et al. (2018); Tellex et al. (2011)). However, a single instruction may be insufficient to fully communicate the full intent of a desired behavior. For example, if we would like a robot to position an object on a table in a particular place, we might find it easier to guide it by telling it which way to move, rather than verbally defining a coordinate in space. Furthermore, an autonomous agent might be unable to deduce how to perform a task from a single instruction, even if it is very precise. In both cases, interactive and iterative corrections can help resolve confusion and ambiguity, and indeed humans often employ corrections when communicating task goals to each other.

In this paper, our goal is to enable an autonomous agent to accept instructions and then iteratively adjust its policy by incorporating interactive corrections. This type of in-the-loop supervision can guide the learner out of local optima, provide fine-grained task definition, and is natural for humans to provide to the agent. As we discuss in Section 2, iterative language corrections can be substantially more informative than simpler forms of supervision, such as preferences, while being substantially easier and more natural to provide than reward functions or demonstrations.

In order to effectively use language corrections, the agent must be able to ground these corrections to concrete behavioral patterns. We propose an end-to-end algorithm for grounding iterative language
corrections by using a multi-task setup to meta-train a model that can ingest its own past behavior and a correction, and then correct its behavior to produce better actions. During a meta-training phase, this model is iteratively retrained on its own behavior (and the corresponding correction) on a wide distribution of known tasks. The model learns to correct the types of mistakes that it actually tends to make in the world, by interpreting the language input. At meta-test time, this model can then generalize to new tasks, and learn those tasks quickly through iterative language corrections.

The main contributions of our work are the formulation of language-guided policy learning (LGPL) via meta-learning, as well as a practical LGPL meta-learning algorithm and model. We evaluate our approach on a simulated task which requires the policy to navigate a complex world with partial observation, seeking out user-specified objects and delivering them to user-specified locations. This domain requires the policy to ground the corrections in terms of objects and places.

2 RELATED WORK

Reward functions (Sutton & Barto, 1998) and demonstrations (Argall et al., 2009) are generally the most common methods for specifying tasks for autonomous agents in sequential decision making problems. Prior work has studied a wide range of different techniques for both imitation learning (Ziebart et al., 2008; Abbeel & Ng, 2004) and reward specification, including methods that combine the two to extract reward functions and goals from user examples (Fu et al., 2018; Thomaz et al., 2006) and demonstrations (Fu et al., 2017; Liu et al., 2017). Other works have proposed modalities such as preferences (Christiano et al., 2017) or numerical scores (Warnell et al., 2017). Natural language presents a particularly appealing modality for task specification, since it enables humans to communicate task goals quickly and easily. Unlike demonstrations, language commands do not require being able to perform the task. Unlike reward functions, language commands do not require any manual engineering. Finally, in comparison to low-bandwidth supervision modalities, such as examples of successful outcomes or preferences, language commands can communicate substantially more information, both about the goals of the task and how it should be performed.

A considerable body of work has sought to ground natural language commands to meaningful behaviors. These works typically use a large supervised corpus in order to learn policies that are conditioned on natural language commands (MacMahon et al., 2006; Branavan et al., 2009; Vogel & Jurafsky, 2010; Chen & Mooney, 2011; Tellex et al., 2011; Artzi & Zettlemoyer, 2013; Kim & Mooney, 2013; Andreas & Klein, 2015; Mirsa et al., 2017; Andreas et al., 2018; Oh et al., 2017). Other works consider using a known reward function in order to learn how to ground language into expert behaviors (Janner et al., 2018; Andreas & Klein, 2015). Most of these works consider the case...
of instruction following. However, tasks can often be quite difficult to specify with a single language description, and may require interactive guidance in order to be achieved. We focus on this setting in our work, where the agent improves its behavior via iterative language corrections.

While the focus in our work is on incorporating language corrections, several prior works have also studied reinforcement learning and related problems with in-the-loop feedback of other forms [Akrou et al., 2011; Pilarski et al., 2011; Akrou et al., 2012; El Asri et al., 2016; Wang et al., 2016b; Warnell et al., 2017]. In contrast to these prior works, we study how to incorporate language corrections, which are more natural for humans to specify and can carry more information about the task. However, language corrections also present the challenge that the agent must learn how to ground them in behavior. To this end, we introduce an end-to-end algorithm that directly associates language with changes in behavior without intermediate supervision about object identities or word definitions.

Our approach to learning to learn from language corrections is based on meta-reinforcement learning. In meta-reinforcement learning, a meta-training procedure is used to learn a procedure (represented by initial network weights or a model that directly ingests past experience) [Schmidhuber, 1987] that can adapt to new tasks at meta-test time. However, while prior work has proposed meta-reinforcement learning for model-free RL [Wang et al., 2016a; Duan et al., 2016; Finn et al., 2017; Mishra et al., 2017], model-based RL [Clavera et al., 2018], and a wide range of supervised tasks [Snell et al., 2017; Santoro et al., 2016; Vinyals et al., 2016; Sung et al., 2017], to our knowledge no prior work has proposed meta-training of policies that can acquire new tasks from iterative language corrections.

3 Problem Formulation

We consider the sequential decision making framework, where an agent observes states $s \in S$, chooses to execute actions $a \in A$ and transitions to a new state $s'$ via the transition dynamics $T(s'|s, a)$. For goal directed agents, the objective is typically to learn a policy $\pi_\theta$ that chooses actions enabling the agent to achieve the desired goal. In this work, the agent’s goal is specified by a language instruction $L$. This instruction describes what the general objective of the task is, but may be insufficient to fully communicate the intent of a desired behavior.

The agent can attempt the task multiple times, and after each attempt, the agent is provided with a language correction. Each attempt results in a trajectory $\tau = (s_0, a_0, s_1, a_1, ..., s_T, a_T)$, the result of the agent executing its policy $\pi_\theta(a|s, L)$ in the environment. After each attempt, the user generates a correction according to some unknown stochastic function of the trajectory $C \sim F_{corr}(C|\tau)$. $C$ is a language phrase that indicates how to improve the current trajectory $\tau$ to bring it closer to accomplishing the goal. This process is repeated for multiple trials, and we will use $\tau_i$ to denote the trajectory on the $i^{th}$ trial, and $C_i$ to denote the corresponding correction. An effective model should be able to incorporate these corrections to come closer to achieving the goal. This process is illustrated in Figure 1.

In the next section, we will describe a model that can incorporate iterative corrections, and then describe a meta-training procedure that can train this model to incorporate corrections effectively.

4 The Language-Guided Policy Learning Model

As described in Section 3, our model for language-guided policy learning (LGPL) must take in an initial language instruction, and then iteratively incorporate corrections after each attempt at the task. This requires the model to ground the contents of the correction in the environment, and also interpret it in the context of its own previous trajectory so as to decide which actions to attempt next. To that end, we propose a deep neural network model, shown in Figure 2, that can accept the instruction, correction, previous trajectory, and state as input. The model consists of three modules: an instruction following module, a correction module, and a policy module.

The instruction following module interprets the initial language instruction. The instructions are provided as a sequence of words. This sequence is converted into a sequence of word-embeddings and then fed into a bi-directional LSTM to generate an instruction embedding vector $z_{im}$. This tensor is fed into the policy module as described below.
Figure 2: The architecture of our model. The instruction module embeds the initial instruction $L$, while the correction modules embed the trajectory $\tau_i$ and correction $C_i$ from each previous trial. The features from these corrections are pooled and provided to the policy, together with the current state $s$ and the embedded initial instruction.

The correction module interprets the previous language correction $C_i$ in the context of the previous trajectory $\tau_i$. The previous trajectory is fed into a recurrent network that yields a single tensor $z_{\text{traj}}$. The correction $C_i$, similar to the language description, is converted into a sequence of word-embeddings which is then fed through a bi-directional LSTM, which generates a single tensor $w_i$. It then computes the mean of all the correction embeddings seen up to this point to create the full correction history tensor $w_{\text{hist}} = \frac{1}{i} \sum_{j=0}^{i} w_j$. These two tensors $z_{\text{traj}}$ and $w_{\text{hist}}$ are concatenated to form the output of the correction module $z_{\text{cm}}$.

The policy module uses the tensors from the instruction following module $z_{\text{im}}$ and the correction module $z_{\text{cm}}$, with the environment state $s$, to decide the correct actions to take. This module inputs $z_{\text{cm}}$, $z_{\text{im}}$ and $s$ and generates an action distribution $p(a|s)$ that determine how the agent should act.

Note that, by iteratively incorporating language corrections, such a model in effect implements a learning algorithm, analogously to meta-reinforcement learning recurrent models proposed in prior work that read in previous trajectories and rewards (Duan et al., 2016; Wang et al., 2016a). However, in contrast to these methods, our model has to use the language correction to improve, essentially implementing an interactive, user-guided reinforcement learning algorithm. As we will demonstrate in our experiments, iterative corrections cause this model to progressively improve its performance on the task. In the next section, we will describe a meta-learning algorithm that can train this model such that it is able to adapt to iterative corrections effectively at meta-test time.

5 Meta-Training the LGPL Model to Learn From Corrections

In order for the LGPL model to be able to learn behaviors from corrections, it must be meta-trained to understand both instructions and corrections properly, put them in the context of its own previous trajectories, and associate them with objects and events in the world. For clarity of terminology, we will use the term “meta-training” to denote the process of training the model, and “meta-testing” to denote its use for solving a new task with language corrections.

During meta-training, we assume access to samples from a distribution of meta-training tasks $T \sim p(T)$. The tasks that the model will be asked to learn at meta-test time are distinct from the meta-training tasks, though we assume them to be drawn from the same distribution, which is analogous to the standard distribution assumption in supervised learning. The tasks have the same state space $S$ and action space $A$, but each task has a distinct goal, and each task $T$ can be described by a different language instruction $L_T$. In general, more than one instruction could describe a single task, and the instructions might contain ambiguity.

Each of the tasks during meta-training also has a ground truth objective, which is provided to the meta-learning either by means of a reward function or by access to an expert (e.g., a human user) who provides near-optimal actions directly. If the task is specified via a reward function, we can
use any existing reinforcement learning algorithm to obtain a solution (a policy), which we can then use to generate near-optimal actions. Therefore, we derive the algorithm for the case where we have access to a near-optimal policy \( \pi^* \) for each task \( T \). In our experiments, \( \pi^* \) is obtained via reinforcement learning from ground truth rewards. For each meta-training task, we also assume that we can sample from the corresponding correction function \( F_{corr,T}(C|\tau) \), which generates a correction \( C \) for the trajectory \( \tau \). In practice, these corrections might be provided by a human annotator, though we use a computational proxy in our experiments.

By using \( \pi^* \), \( L_T \), and \( F_{corr,T}(\tau) \), we can train our model for each task by using a variant of the DAgger algorithm [Ross et al., 2011], which was originally proposed for single-task imitation learning, where a learner policy is trained to mimic a near-optimal expert. We extend this approach to the setting of meta-learning, where we use it to meta-train the LGPL model. Starting from an initialization where the previous trajectory \( \tau_0 \) and \( C_0 \) are set to be empty sequences, we repeat the following process: first, we run the policy corresponding to the current learned model \( \pi(a|s, L_T, \tau_0, C_0) \) to generate a new trajectory \( \tau_1 \) for the task \( T \). Every state along \( \tau_1 \) is then labeled with near-optimal actions by using \( \pi^* \) to produce a set of training tuples \( (L_T, \tau_0, C_0, a^*) \). These tuples are appended to the training set \( D \). Then, a correction \( C_1 \) is sampled from \( F_{corr,T}(C|\tau) \), and a new trajectory is sampled from \( \pi(a|s, L_T, \tau_1, C_1) \). This trajectory is again labeled by the expert and appended to the dataset. In the same way, we iteratively populate the training set \( D \) with the states, corrections, and prior trajectories observed by the model, all labeled with near-optimal actions. This process is repeated for a fixed number of corrections or until the task is solved, for each of the meta-training tasks. The model is then trained via supervised maximum likelihood learning on all of the samples in the dataset \( D \). Then, following the DAgger algorithm, the updated policy is used to again collect data for each of the tasks, which is appended to the dataset and used to train the policy again, until the process converges or a fixed number of iterations. This algorithm is summarized in Algorithm 1 and Figure 3.

6 LEARNING NEW TASKS WITH THE LGPL MODEL

Using a LGPL model meta-trained as described in the previous section, we can solve new "meta-testing" tasks \( T_{test} \sim p(T) \) drawn from the same distribution of tasks with interactive language corrections. An initial instruction \( L_T \) is first provided by the user, and the procedure for adapting with corrections follows the illustration in Figure 1. The learned policy is initially rolled out in the environment conditioned on \( L_T \), and with the previous trajectory \( \tau_0 \) and correction \( C_0 \) initialized to the empty sequence. Once this policy generates a trajectory \( \tau_1 \), we can use the correction function \( F_{corr,T} \) to generate a correction \( C_1 = F_{corr,T}(\tau_1) \). The trajectory \( \tau_1 \), along with the correction \( C_1 \), gives us a new improved policy which is conditioned on \( L_T, \tau_1, C_1 \). This policy can be executed in the environment to generate a new trajectory \( \tau_2 \), and the process repeats until convergence, thereby.
learning the new task. We provide the policy with the previous corrections as well but omit in the notation for clarity. This procedure is summarized in Algorithm 2.

This procedure is reminiscent of meta-reinforcement learning ([Finn et al., 2017] [Duan et al., 2016], but uses grounded natural language corrections in order to guide learning of new tasks with feedback provided in the loop. The advantage of such a procedure is that we can iteratively refine behaviors quickly for tasks that are hard to describe with high level descriptions. Additionally, providing language feedback iteratively in the loop may reduce the overall amount of supervision needed to learn new tasks. Using easily available natural language corrections in the loop can change behaviors much more quickly than scalar reward functions.

### Algorithm 1: LGPL meta-training algorithm.

1. Initialize data buffer $D$.
2. for iteration $j$ do
   1. for task $T$ do
      1. Initialize $\tau_0 = 0$ and $C_0 = 0$
      2. for corr iter $i \in \{0, \ldots, c_{\text{max}}\}$ do
         1. Execute $\pi(a|s, L_T, \tau_i, C_i)$ on $T$ to collect $\tau_{i+1}$
         2. Obtain $C_{i+1} \sim F_{\text{corr}, T}(\tau_{i+1})$
         3. Label $a^* \sim \pi^*_T(a|s), \forall s \in \tau_{i+1}$
         4. Add $(L_T, \tau_i, C_i, s, a^*), \forall s \in \tau_{i+1}$ to $D$
      3. Train $\pi$ on $D$.

### Algorithm 2: Meta-testing: learning new tasks with the LGPL model.

1. Given new task $T_i$, with instruction $L_T$
2. Initialize $\tau_0 = 0$ and $C_0 = 0$
3. for corr iter $i \in \{0, \ldots, c_{\text{max}}\}$ do
   1. Execute $\pi(a|s, L_T, \tau_i, C_i)$ on $T$ to collect $\tau_{i+1}$
   2. Obtain $C_{i+1} \sim F_{\text{corr}, T}(\tau_{i+1})$

7 Experiments

Our experiments aim to analyze LPGL as a technique for carrying out varied goals in a partially observed simulated environment. The first goal of our evaluation is to understand where LGPL can benefit from iterative corrections – that is, does the policy’s ability to succeed at the task improve as each new correction provided. We then evaluate our method comparatively, in order to understand whether iterative corrections provide an improvement over standard instruction-following methods, and also compare LGPL to an oracle model that receives a much more detailed instruction, but without the iterative structure of interactive corrections. Our code and supplementary material will be available at [https://sites.google.com/view/lgpl/home](https://sites.google.com/view/lgpl/home).

7.1 Experimental Setup

Our experimental evaluation involves a discrete environment that represents the floor-plan of a building with six rooms (see Figure 3), based on an environment from Chevalier-Boisvert & Willems (2018). Each of the six rooms has a uniquely colored door that must be opened to enter the room, and some rooms contain objects with different colors and shapes. The environment has colored floor tiles, one of which is the true goal, with the others serving as decoy goals. The actions allow for moving in each of the cardinal directions, as well as two additional actions to pick up and drop objects. The environment is partially observed: the policy only observes the contents of an ego-centric 7 by 7 region centered on the present location of the agent, and does not permit seeing through walls or closed doors, which means that the contents of the room can only be observed by first opening the door leading to that room. The task given to the agent consists of two phases: first, the agent must navigate to the goal object, which is in a closed room, and pick it up; then, the agent must bring the goal object to the goal square which is in a different closed room.

This environment was selected due to its structure, which causes the natural language instruction associated with a task, detailed in the next section, to be underspecified. In particular, due to the partial observability of this environment, the agent receives no information corresponding to either the location of the goal object or goal square in its initial observation. Additionally, the doors and walls make it so that proximity to the objects is insufficient to reveal them to the agent. This structure lends itself to the use of corrections, which can guide the agent to the appropriate doors, revealing the information it needs to solve the task.
Environments are generated by sampling a goal object color, goal object shape, and goal square color which are placed at random locations in different random rooms. There are 6 possible colors and 3 possible object shapes. Decoy objects and goals are placed among the six rooms at random locations. The only shared aspect between tasks are the color of the doors so the agent must learn to generalize across a variety of different objects across different locations.

7.2 Language Instructions and Corrections

The instruction is given as "Move <goal object color> <goal object shape> to <goal square color> square" which does not provide information as to how to accomplish the task. Since the agent cannot see inside closed rooms, the agent does not initially know the locations of the goal object or square, requiring it either to explore or rely on external information.

To generate the corrections, we describe a task as a list of subgoals that the agent must complete. For example, the instruction in Figure 5 is "Move green triangle to green square", and the subgoals are "enter the blue room", "pick up the green triangle", "exit the blue room", "enter the purple room", and "go to the green goal". The correction for a given trajectory is then the first subgoal that the agent failed to complete. The multistep nature of this task also makes it challenging, as the agent must remember to solve previously completed subgoals while incorporating the corrections to solve the next subgoal.

7.3 Comparisons

We compare our method to an instruction following method and a method that receives full information. Both baselines are also trained with DAgger. The instruction following baseline only receives the instruction, which is ambiguous and does not contain the location of the goal object and square. The full information baseline receives all the subgoals that are needed to solve the task, but does not receive them interactively. We measure the performance of an agent on the task by computing the completion rate: the fraction of subgoals that the agent has successfully completed. The maximum number of subgoals is always 5. We expect our model to perform better than the instruction following baseline as it can receive the missing information through the corrections. We also expect to perform as well as or better than the full information baseline with fewer than 5 corrections since in many cases our model will then receive all of the subgoals.

Figure 6: Example task with corrections. Instruction: The agent receives the initial instruction. Correction 1: The agent mistakenly goes into the gray door, so it receives the correction to enter the green room, where the purple ball is located. Correction 2: The agent successfully picks up the ball, but then mistakenly enters the blue room, so it receives the correction to enter the red room, where the goal is located. Correction 3: The agent brings the object to the goal and solves the task.
7.4 Learning New Tasks Quickly with Language Corrections

As described above, we consider a multi-task setting in the object relocation domain where the task is to pick up a particular object in a room and bring it to a particular goal location in a different room. The test tasks consist of new configurations of objects and goals.

We measure the completion rate of our method for various numbers of corrections on the training and test tasks. The instruction baseline does not have enough information and is unable to effectively solve the task. As expected, we see increasing completion rates as the number of corrections increases and the agent incrementally gets further in the task. Our method matches close to the full information baseline with 3 corrections and outperforms it with 4 or more corrections. Since the full information baseline receives all 5 subgoals, this means our method performs better with less information. The interactive nature of our method allows it to receive only the information it needs to solve the task. In many cases where the agent succeeds we notice that agent only needs 2 corrections where the first correction is the location of the goal object and the second correction is the location of the goal square (Figure 6). Furthermore, our model must learn to map corrections to changes in behavior which may be more modular, disentangled, and easier to generalize compared to mapping a long list of instructions to a single policy that can solve the task.

<table>
<thead>
<tr>
<th>Method</th>
<th>Instruction</th>
<th>Full Information</th>
<th>C₀</th>
<th>C₁</th>
<th>C₂</th>
<th>C₃</th>
<th>C₄</th>
<th>C₅</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train Tasks</td>
<td>0.094</td>
<td>0.79</td>
<td>0.08</td>
<td>0.49</td>
<td>0.70</td>
<td>0.80</td>
<td>0.85</td>
<td>0.88</td>
</tr>
<tr>
<td>Test Tasks</td>
<td>0.091</td>
<td>0.72</td>
<td>0.057</td>
<td>0.44</td>
<td>0.60</td>
<td>0.68</td>
<td>0.74</td>
<td>0.77</td>
</tr>
</tbody>
</table>

Table 1: Completion rates on training and test tasks for baseline methods and ours. Cᵢ denotes that the agent has received i corrections. LGPL is able to quickly incorporate corrections to improve agent behavior over instruction following with fewer corrections than full information.

7.5 Analyzing Behavior of LGPL

We perform ablations to analyze the importance of each component of our model in Figure 2. For the three ablations, we remove the instruction L, remove the previous trajectory τᵢ, and provide only the immediate correction Cᵢ instead of all previous corrections. We find that removing the instruction hurts the performance the least. This makes sense because the model can receive the information contained in the instruction through the corrections. Removing the previous corrections hurts the performance the most. During failure cases we notice that the agent forgets what it had done previously and erases the progress it made. This explains the dip in performance from C₁ to C₂.

<table>
<thead>
<tr>
<th>Ablations</th>
<th>C₀</th>
<th>C₁</th>
<th>C₂</th>
<th>C₃</th>
<th>C₄</th>
<th>C₅</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base</td>
<td>0.057</td>
<td>0.44</td>
<td>0.60</td>
<td>0.68</td>
<td>0.74</td>
<td>0.77</td>
</tr>
<tr>
<td>No L</td>
<td>0.059</td>
<td>0.43</td>
<td>0.58</td>
<td>0.67</td>
<td>0.72</td>
<td>0.72</td>
</tr>
<tr>
<td>No τ</td>
<td>0.061</td>
<td>0.41</td>
<td>0.58</td>
<td>0.65</td>
<td>0.67</td>
<td>0.70</td>
</tr>
<tr>
<td>Only C₁</td>
<td>0.059</td>
<td>0.42</td>
<td>0.37</td>
<td>0.49</td>
<td>0.46</td>
<td>0.53</td>
</tr>
</tbody>
</table>

Table 2: Ablation Experiments analyzing the importance of various components of the model. We see that removing previous corrections (only C₁) performs the worst, while removing instruction L is less impactful.

7.6 Discussion and Future Work

We presented language-guided policy learning (LGPL), a framework for interactive learning of tasks with in-the-loop language corrections. In LGPL, the policy attempts successive trials in the environment, and receives language corrections that suggest how to improve the next trial over the previous one. The LGPL model is trained via meta-learning, using a dataset of other tasks to learn how to ground language corrections in terms of behaviors and objects. While our method is amenable to natural language corrections and instructions, an exciting direction for future work would be to incorporate real human annotations into the training process. Finally, in order to scale corrections to real-world tasks, it will be vital to handle new concepts, in terms of actions or objects, not seen at training time. Two such ways one could approach handling these new concepts could be innovations at the model level, such as using meta-learning, or at the interface level, allowing humans to describe or point to new objects to help the agent identify them.
REFERENCES


A Appendix

A.1 Visualizing Behavior

Figure 7: Failure example. The orange arrow shows the task, the white arrows show the net trajectory.

It is possible to visualize failure cases, which illuminate the behavior of the algorithm on challenging tasks. In the failure case in Figure 7, we note that the agent is able to successfully enter the purple room, pickup the green ball, and exit. However, after it receives the fourth correction telling it to go to the green goal, it forgets to pick up the green ball.

Figure 8: Success example. The orange arrow shows the task, the white arrows show the net trajectory.

Additionally, we present a success case in Figure 8, where the agent successfully learns to solve the task through iterative corrections, making further progress in each frame.