META-LEARNING LANGUAGE-GUIDED POLICY LEARNING

Anonymous authors
Paper under double-blind review

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

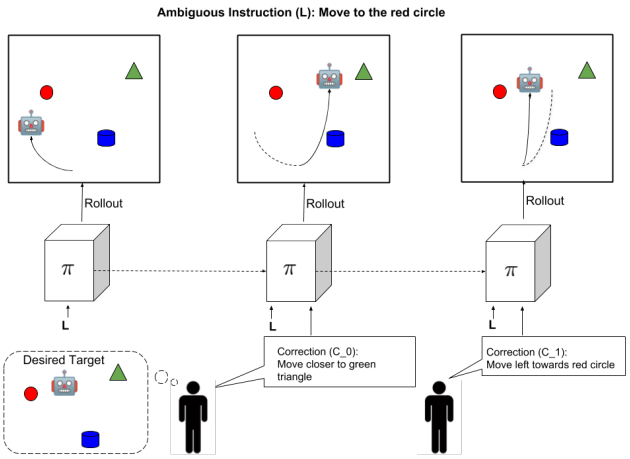

Figure 1: An example where corrections disambiguate an instruction. The agent is unable to fully deduce the user's intent from the instruction alone and iterative language corrections guide the agent to the correct position. Our method is concerned with meta-learning policies that can ground language corrections in their environment and use them to improve through iterative feedback.

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 Akrour et al. (2011); Pilarski et al. (2011); Akrour 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 $s \in \mathcal{S}$, chooses to execute actions $a \in \mathcal{A}$ and transitions to a new state s' via the transition dynamics $\mathcal{T}(s'|s, a)$. For goal directed agents, the objective is typically to learn a policy π_θ 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 τ to bring it closer to accomplishing the goal. This process is repeated for multiple trials, and we will use τ_i to denote the trajectory on the ith 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 as 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.

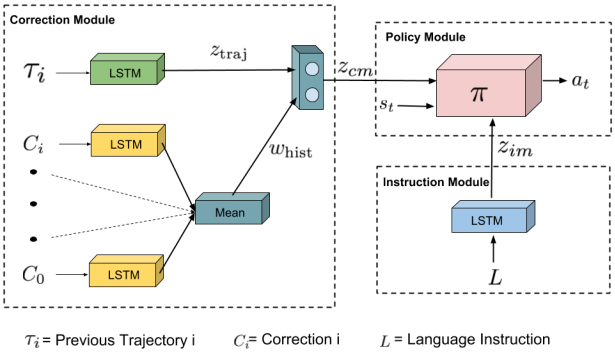

Figure 2: The architecture of our model. The instruction module embeds the initial instruction L, while the correction modules embed the trajectory τ_i and correction C_i from each previous trial. The features from these corrections are pooled and provided to the policy, together with the current state s and the embedded initial instruction.

The correction module interprets the previous language correction C_i in the context of the previous trajectory τ_i. The previous trajectory is fed into a recurrent network that yields a single tensor z_{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_{traj} and w_{hist} are concatenated to form the output of the correction module z_{cm}.

The policy module uses the tensors from the instruction following module z_{im} and the correction module z_{cm}, with the environment state s, to decide the correct actions to take. This module inputs z_{cm}, z_{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 $\mathcal{S}$ and action space $\mathcal{A}$, 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

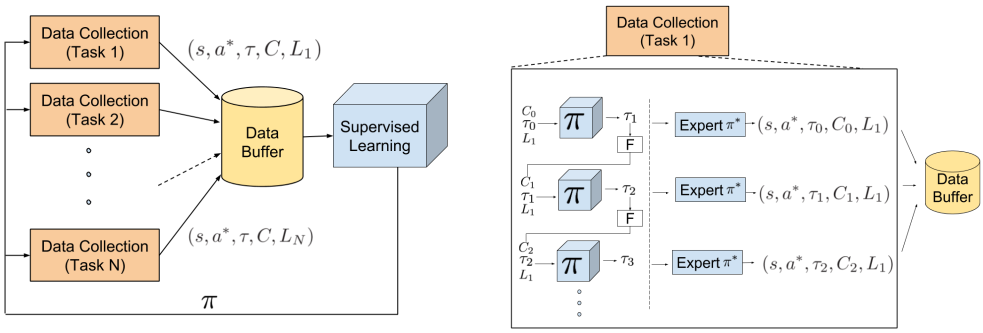

Figure 3: **Left:** Overall training paradigm. We collect data for each task $[1, 2, \ldots, N]$ via a DAgger like procedure, and put this data from all tasks into a data buffer. This is used to train a LGPL policy with supervised learning. The trained policy is then used to collect data for individual tasks again, repeating the process till convergence. **Right:** Individual data collection paradigm for a single task. The LGPL policy is initially executed to obtain a trajectory τ_1. This trajectory is corrected by an expert π^* to generate data to be added to the buffer. The trajectory is then used to generate a correction, which is fed back into π, along with τ_1 to generate a new trajectory τ_2. This repeats until a maximum number of corrections are given, adding data to the buffer at each step.

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 $\pi^*_T(a|s)$ for each task T. In our experiments, $\pi^*_T(a|s)$ 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 τ. In practice, these corrections might be provided by a human annotator, though we use a computational proxy in our experiments.

By using $\pi^*_T(a|s)$, 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 τ_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 τ_1 for the task T. Every state along τ_1 is then labeled with near-optimal actions by using $\pi^*_T(a|s)$ to produce a set of training tuples $(L_T, \tau_0, C_0, s, a^*)$. 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 τ_0 and correction C_0 initialized to the empty sequence. Once this policy generates a trajectory τ_1, we can use the correction function $F_{corr,T}$ to generate a correction $C_1 = F_{corr,T}(\tau_1)$. The trajectory τ_1, along with the correction C_1 gives us a new improved policy which is conditioned on L_T, τ_1, and C_1. This policy can be executed in the environment to generate a new trajectory τ_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.

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

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

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

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`.

7.1 EXPERIMENTAL SETUP

Our experimental evaluation involves a discrete environment that represents the floor-plan of a building with six rooms (see Figure 4), 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.

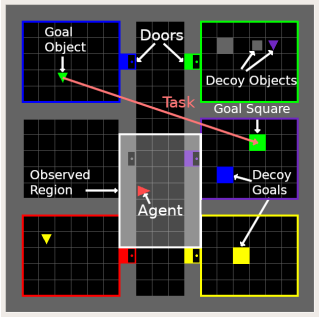

Figure 4: The multiroom object manipulation environment with labeled components.

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

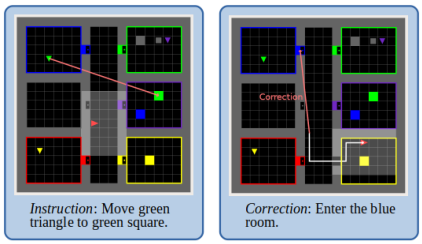

Figure 5: Left: Environment with task shown by orange arrow. Right: Correction shown by orange arrow.

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.

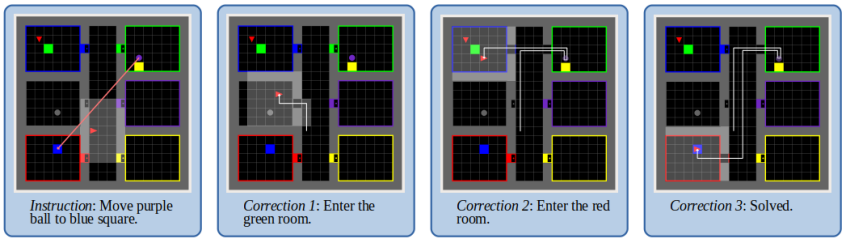

Figure 6: Example task with corrections. Instruction: The agent receives the initial instruction. Correction 1: The agent mistakenly goes into the gray door, so it receives the correction to enter the green room, where the purple ball is located. Correction 2: The agent successfully picks up the ball, 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.

Method	Instruction	Full Information	C_0	C_1	C_2	C_3	C_4	C_5
Train Tasks	0.094	0.79	0.08	0.49	0.70	0.80	0.85	0.88
Test Tasks	0.091	0.72	0.057	0.44	0.60	0.68	0.74	0.77

Table 1: Completion rates on training and test tasks for baseline methods and ours. C_i 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 τ_i, and provide only the immediate correction C_i 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 C_1 to C_2.

Ablations	C_0	C_1	C_2	C_3	C_4	C_5
Base	0.057	0.44	0.60	0.68	0.74	0.77
No L	0.059	0.43	0.58	0.67	0.72	0.72
No τ	0.061	0.41	0.58	0.65	0.67	0.70
Only C_i	0.059	0.42	0.37	0.49	0.46	0.53

Table 2: Ablation Experiments analyzing the importance of various components of the model. We see that removing previous corrections (only C_i) 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

Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Machine Learning, Proceedings of the Twenty-first International Conference (ICML 2004), Banff, Alberta, Canada, July 4-8, 2004*, 2004. doi: 10.1145/1015330.1015430.

Riad Akrour, Marc Schoenauer, and Michèle Sebag. Preference-based policy learning. In *ECML/PKDD*, 2011.

Riad Akrour, Marc Schoenauer, and Michèle Sebag. APRIL: active preference-learning based reinforcement learning. *CoRR*, abs/1208.0984, 2012.

Jacob Andreas and Dan Klein. Alignment-based compositional semantics for instruction following. In *EMNLP*, 2015.

Jacob Andreas, Dan Klein, and Sergey Levine. Learning with latent language. In *NAACL*, 2018.

Brenna D. Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robot. Auton. Syst.*, 57(5):469–483, May 2009. ISSN 0921-8890. doi: 10.1016/j.robot.2008.10.024.

Yoav Artzi and Luke Zettlemoyer. Weakly supervised learning of semantic parsers for mapping instructions to actions. In *TACL*, 2013.

SRK Branavan, Harr Chen, Luke Zettlemoyer, and Regina Barzilay. Reinforcement learning for mapping instructions to actions. In *ACL*, 2009.

David Chen and Raymond Mooney. Learning to interpret natural language navigation instructions from observations. In *AAAI*, 2011.

Maxime Chevalier-Boisvert and Lucas Willems. Minimalistic gridworld environment for openai gym. `https://github.com/maximecb/gym-minigrid`, 2018.

Paul F. Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pp. 4302–4310, 2017.

Ignasi Clavera, Anusha Nagabandi, Ronald S. Fearing, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Learning to adapt: Meta-learning for model-based control. *CoRR*, abs/1803.11347, 2018.

Yan Duan, John Schulman, Xi Chen, Peter L. Bartlett, Ilya Sutskever, and Pieter Abbeel. Rl^2: Fast reinforcement learning via slow reinforcement learning. *CoRR*, abs/1611.02779, 2016.

Layla El Asri, Jing He, and Kaheer Suleman. A sequence-to-sequence model for user simulation in spoken dialogue systems. In *arXiv preprint arXiv:1607.00070*, 2016.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pp. 1126–1135, 2017.

Daniel Fried, Jacob Andreas, and Dan Klein. Unified pragmatic models for generating and following instructions. In *NAACL*, 2018.

Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adversarial inverse reinforcement learning. *CoRR*, abs/1710.11248, 2017.

Justin Fu, Avi Singh, Dibya Ghosh, Larry Yang, and Sergey Levine. Variational inverse control with events: A general framework for data-driven reward definition. *CoRR*, abs/1805.11686, 2018.

Michael Janner, Karthik Narasimhan, and Regina Barzilay. Representation learning for grounded spatial reasoning. In *ACL*, 2018.

Joohyun Kim and Raymond Mooney. Adapting discriminative reranking to grounded language learning. In *ACL*, 2013.

Yuxuan Liu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Imitation from observation: Learning to imitate behaviors from raw video via context translation. *CoRR*, abs/1707.03374, 2017.

Matt MacMahon, Brian Stankiewicz, and Benjamin Kuipers. Walk the talk: Connecting language, knowledge, and action in route instructions. In *AAAI*, 2006.

Dipendra Mirsa, John Langford, and Yoav Artzi. Mapping instructions and visual observations to actions with reinforcement learning. In *EMNLP*, 2017.

Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. Meta-learning with temporal convolutions. *CoRR*, abs/1707.03141, 2017.

Junhyuk Oh, Satinder P. Singh, Honglak Lee, and Pushmeet Kohli. Zero-shot task generalization with multi-task deep reinforcement learning. *CoRR*, abs/1706.05064, 2017.

Patrick Pilarski, Michael Dawson, Thomas Degris, Farbod Fahimi, Jason Carey, and Richard Sutton. Online human training of a myoelectric prosthesis controller via actor-critic reinforcement learning. In *IEEE ICORR*, 2011.

Stéphane Ross, Geoffrey J. Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011*, pp. 627–635, 2011.

Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy P. Lillicrap. Meta-learning with memory-augmented neural networks. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pp. 1842–1850, 2016.

Jürgen Schmidhuber. *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook*. PhD thesis, Technische Universität München, 1987.

Jake Snell, Kevin Swersky, and Richard S. Zemel. Prototypical networks for few-shot learning. *CoRR*, abs/1703.05175, 2017.

Flood Sung, Li Zhang, Tao Xiang, Timothy M. Hospedales, and Yongxin Yang. Learning to learn: Meta-critic networks for sample efficient learning. *CoRR*, abs/1706.09529, 2017.

Richard Sutton and Andrew Barto. *Introduction to Reinforcement Learning*, volume 1. MIT Press, 1998.

Stephanie Tellex, Thomas Kollar, Steven Dickerson, Matthew Walter, Ashis Banerjee, Steph Teller, and Nicholas Roy. Understanding natural language commands for robotic navigation and mobile manipulation. In *AAAI*, 2011.

Andrea Lockerd Thomaz, Guy Hoffman, and Cynthia Breazeal. Reinforcement learning with human teachers: Understanding how people want to teach robots. In *The 15th IEEE International Symposium on Robot and Human Interactive Communication, RO-MAN 2006, Hatfield, Herthfordshire, UK, 6-8 September, 2006*, pp. 352–357, 2006. doi: 10.1109/ROMAN.2006.314459.

Oriol Vinyals, Charles Blundell, Tim Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pp. 3630–3638, 2016.

Adam Vogel and Dan Jurafsky. Learning to follow navigational directions. In *Association for Computational Linguistics (ACL)*, 2010.

Jane X. Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z. Leibo, Rémi Munos, Charles Blundell, Dharshan Kumaran, and Matthew Botvinick. Learning to reinforcement learn. *CoRR*, abs/1611.05763, 2016a.

Sida Wang, Percy Liang, and Chris Manning. Learning language games through interaction. In *ACL*, 2016b.

Garrett Warnell, Nicholas R. Waytowich, Vernon Lawhern, and Peter Stone. Deep TAMER: interactive agent shaping in high-dimensional state spaces. *CoRR*, abs/1709.10163, 2017.

Brian D. Ziebart, Andrew L. Maas, J. Andrew Bagnell, and Anind K. Dey. Maximum entropy inverse reinforcement learning. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, pp. 1433–1438, 2008.

A APPENDIX

A.1 VISUALIZING BEHAVIOR

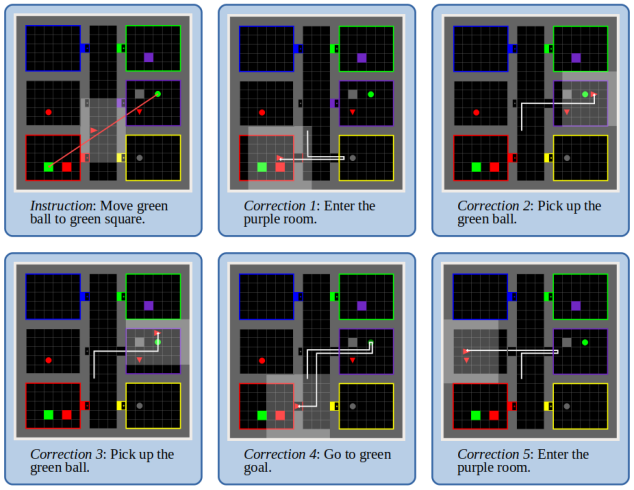

Figure 7: Failure example. The orange arrow shows the task, the white arrows show the net trajectory.

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.

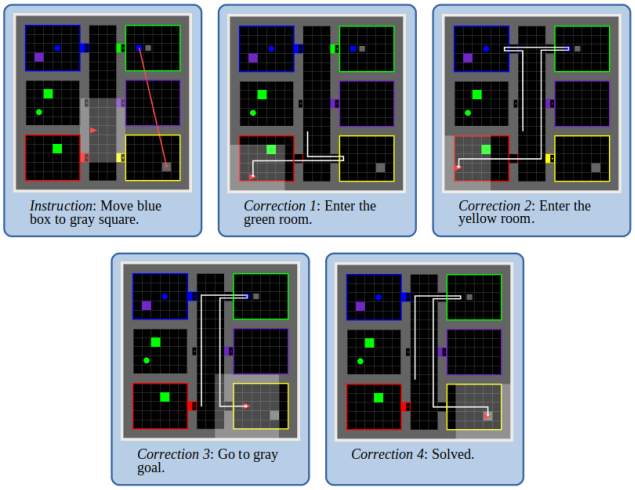

Figure 8: Success example. The orange arrow shows the task, the white arrows show the net trajectory.

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.