Guiding Policies with Language via Meta-Learning

John D. Co-Reyes Abhishek Gupta Suvansh Sanjeev Nick Altieri
John DeNero Pieter Abbeel Sergey Levine
University of California Berkeley

1 Introduction

Behavioral skills or policies for autonomous agents are typically specified in terms of reward functions (in the case of reinforcement learning) (Sutton & Barto, 1998) or demonstrations (in the case of imitation learning) (Argall et al., 2009). However, reward functions must be engineered manually, which can be challenging in real-world environments. 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. 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.

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. 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 The Languaged-Guided Policy Learning Model

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 $\mathcal{T}(s'|s, a)$. 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.

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

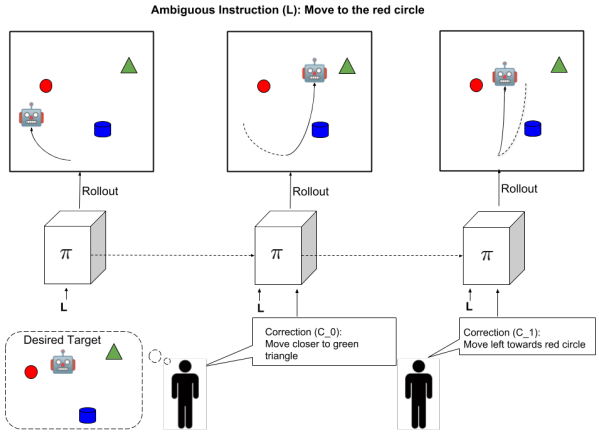

Figure 1: An example where corrections disambiguate an instruction. The agent is unable to fully deduce the user's intent from the instruction alone and iterative language corrections guide the agent to the correct position. Our method is concerned with meta-learning policies that can ground language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

The agent can attempt the task multiple times, and after each attempt, the agent is provided with a language *correction*, shown in Figure 1. 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. τ_i wil denote the trajectory on the i^{th} trial, and C_i will denote the corresponding correction.

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 5, that can accept the instruction, correction, previous trajectory, and state as input.

3 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. During meta-training, we have access to a distribution of meta-training tasks $T \sim p(T)$. Each task has a distinct goal, and each task T can be described by a different language instruction L_T which may be ambiguous.

We assume access to a near-optimal policy $\pi^*_T(a|s)$ for each task T which is obtained via RL 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). We extend this approach to the setting of meta-learning, where we use it to meta-train the LGPL model. Starting with an empty previous trajectory τ_0 and correction C_0, we repeat the following process: first, we run the policy $\pi(a|s, L_T, \tau_0, C_0)$ to generate a new trajectory τ_1 on task T. τ_1 is then labeled with near-optimal actions from $\pi^*_T(a|s)$ to produce training tuples $(L_T, \tau_0, C_0, s, a^*)$ which are added 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. The model is then trained via supervised maximum likelihood learning on D. Then, following the DAgger algorithm, the updated policy is used to collect more data for each of of the tasks which is used to train the policy further. This is summarized in Algorithm 1 and Figure 6.

Using a meta-trained LGPL model, 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 generates 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.

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 $C_{i+1} \sim F_{corr,T}(\tau_{i+1})$

4 Experiments

Our experiments aim to analyze LPGL as a technique for carrying out varied goals in a partially observed simulated environment. We first want to understand where LGPL can benefit from iterative corrections – does the policy's ability to succeed 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`.

Our experimental evaluation involves a discrete environment that represents the floor-plan of a building with six rooms (Figure 2). Each room has a uniquely colored door and contain objects with different colors and shapes. Actions are discrete and involve moving and picking up and dropping objects. The environment is partially observed: the policy only observes an ego-centric 7 by 7 region centered and does not see through walls or closed doors. 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.

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.

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. To generate the corrections, we describe a task as a list of subgoals that the agent must complete. For example, the instruction in Figure 3 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.

4.1 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.

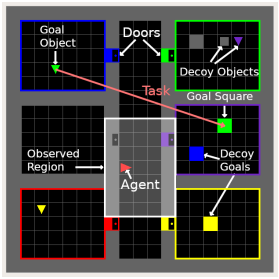

Figure 2: The multiroom object manipulation environment.

4.2 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 4). 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.

5 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.

References

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

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.

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

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

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.

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.

A Appendix

A.1 Environment

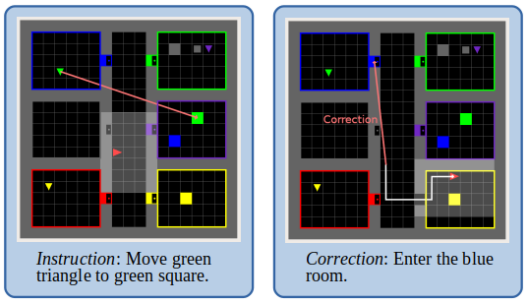

Figure 3: Left: Environment with task shown by orange arrow. Right: Correction shown by orange arrow.

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.

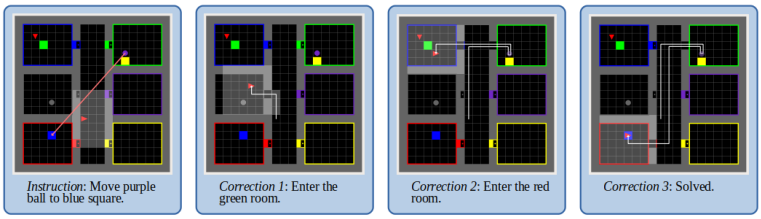

Figure 4: Example task with corrections. Instruction: The agent receives the initial instruction. Correction 1: The agent mistakenly goes into the gray door, so it receives the correction to enter the green room, where the purple ball is located. Correction 2: The agent successfully picks up the ball, 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.

A.2 Training Paradigm

The instruction following module interprets the initial language instruction which is 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}.

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 to generate a correction embedding 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 and transformed by a MLP to 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.

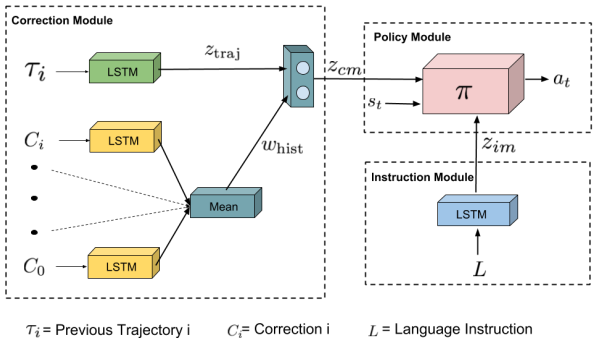

Figure 5: The architecture of our model. The instruction module embeds the initial instruction L, while the correction modules embed the trajectory τ_i and correction C_i from each previous trial. The features from these corrections are pooled and provided to the policy, together with the current state s and the embedded initial instruction.

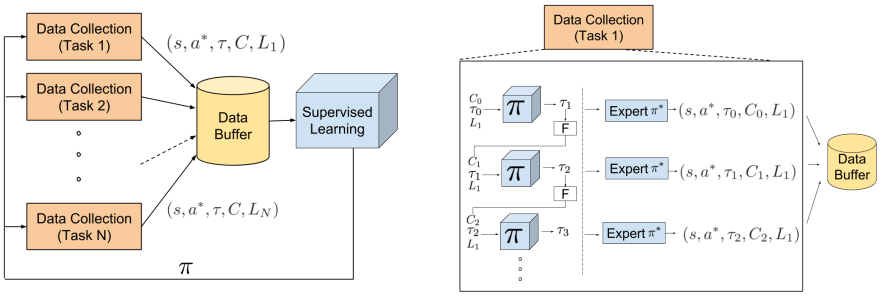

Figure 6: **Left:** Overall training paradigm. We collect data for each task $[1, 2, \ldots, N]$ via a DAgger like procedure, and put this data from all tasks into a data buffer. This is used to train a LGPL policy with supervised learning. The trained policy is then used to collect data for individual 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.

A.3 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.

A.4 Visualizing Behavior

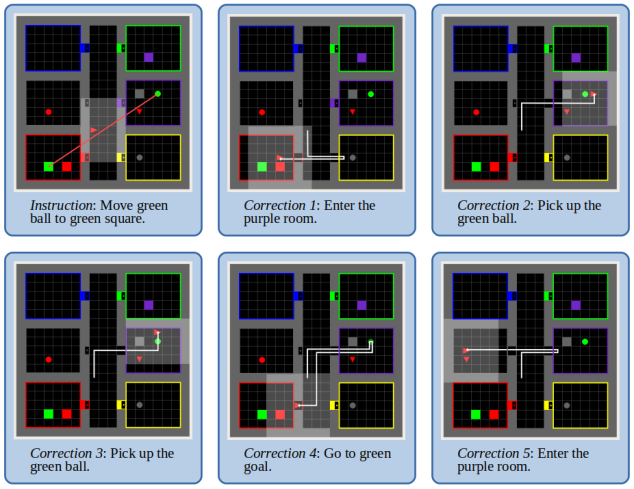

Figure 7: Failure example. The orange arrow shows the task, the white arrows show the net trajectory.

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.

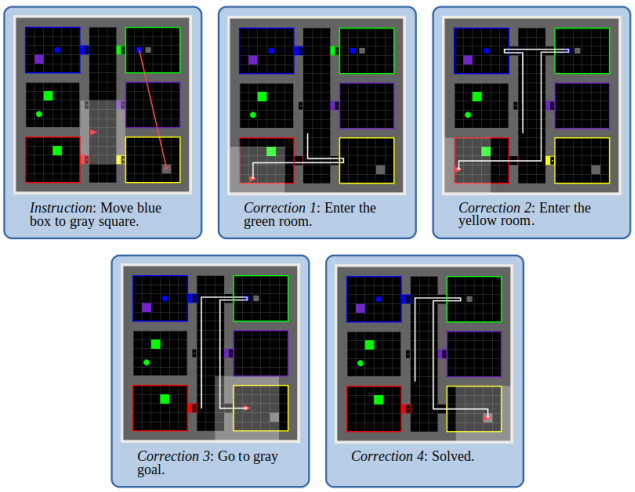

Figure 8: Success example. The orange arrow shows the task, the white arrows show the net trajectory.

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.