In Proceedings of the 28th International Joint Conference on Artificial Intelligence, Macao, China, August 2019.

Using Natural Language for Reward Shaping in Reinforcement Learning

Prasoon Goyal, **Scott Niekum**, **Raymond J. Mooney**
Department of Computer Science
The University of Texas at Austin
{pgoyal, sniekum, mooney}@cs.utexas.edu

Abstract

Recent reinforcement learning (RL) approaches have shown strong performance in complex domains such as Atari games, but are often highly sample inefficient. A common approach to reduce interaction time with the environment is to use reward shaping, which involves carefully designing reward functions that provide the agent intermediate rewards for progress towards the goal. However, designing appropriate shaping rewards is known to be difficult as well as time-consuming. In this work, we address this problem by using natural language instructions to perform reward shaping. We propose the LanguagE-Action Reward Network (LEARN), a framework that maps free-form natural language instructions to intermediate rewards based on actions taken by the agent. These intermediate language-based rewards can seamlessly be integrated into any standard reinforcement learning algorithm. We experiment with Montezuma's Revenge from the Atari Learning Environment, a popular benchmark in RL. Our experiments on a diverse set of 15 tasks demonstrate that, for the same number of interactions with the environment, language-based rewards lead to successful completion of the task 60% more often on average, compared to learning without language.

1 Introduction

Reinforcement learning (RL) has enjoyed much recent success in domains ranging from game-playing to real robotics tasks. However, to make reinforcement learning useful for large-scale real-world applications, it is critical to be able to design reward functions that accurately and efficiently describe tasks. For the sake of simplicity, a common strategy is to provide the agent with sparse rewards—for example, positive reward upon reaching a goal state, and zero reward otherwise. However, it is well-known that learning is often difficult and slow in sparse reward settings [Večerík *et al.*, 2017]. By contrast, dense rewards can be easier to learn from, but are significantly more difficult to specify. In this work, we address this issue by using natural language to provide dense rewards to RL agents in a manner that is easy to specify.

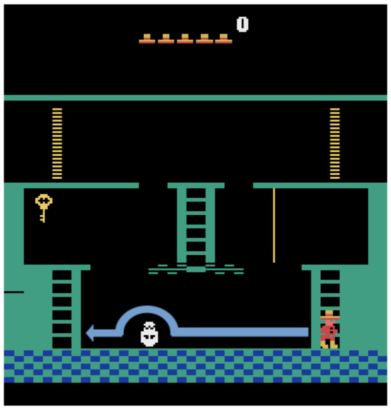


Figure 1: An agent exploring randomly to complete the task described by the blue trajectory may need considerable amount of time to learn the behavior. By giving natural language instructions like "Jump over the skull while going to the left", we can give intermediate signals to the agent for faster learning.

Consider the scenario in Figure 1 from the Atari game Montezuma's Revenge. Suppose we want the agent to go to the left while jumping over the skull (as shown in the blue trajectory). If the agent is given a positive reward only when it reaches the end of the desired trajectory, it may need to spend a significant amount of time exploring the environment to learn that behavior. Giving the agent intermediate rewards for progress towards the goal can help, a technique known as "reward shaping" [Ng *et al.*, 1999]. However, designing intermediate rewards is hard, particularly for non-experts.

Instead, we propose giving the agent intermediate rewards using instructions in natural language. For instance, the agent can be given the following instruction:"Jump over the skull while going to the left" to provide intermediate rewards that accelerate learning. Since natural language instructions can easily be provided even by non-experts, it will enable them to teach RL agents new skills more conveniently.

The main contribution of this work is a new framework which takes arbitrary natural language instruction and the trajectory executed by the agent so far, and makes a prediction

Supplementary material link: https://arxiv.org/abs/1903.02020 whether the agent is following the instruction, which can then be used as an intermediate reward. Our experiments show that by using such reward functions, we can speed up learning in sparse reward settings by guiding the exploration of the agent.

Using arbitrary natural language statements within reinforcement learning presents several challenges. First, a mapping between language and objects/actions must implicitly or explicitly be learned, a problem known as *symbol grounding* [Harnad, 1990]. For example, to make use of the instruction, "Jump over the snake", the system must be able to ground "snake" to appropriate pixels in the current state (assuming the state is represented as an image) and "jump" to the appropriate action in the action space. Second, natural language instructions are often incomplete. For instance, it is possible that the agent is not directly next to the snake and must walk towards it before jumping. Third, natural language inherently involves ambiguity and variation. This could be due to different ways of referring to the objects/actions (e.g. "jump" vs. "hop"), different amounts of information in the instructions (e.g. "Jump over the snake" vs. "Climb down the ladder after jumping over the snake"), or the level of abstraction at which the instructions are given (e.g. a high-level subgoal: "Collect the key" vs. low-level instructions: "Jump over the obstacle. Climb up the ladder and jump to collect the key.")

Once an instruction has been interpreted, we incorporate it into the RL system as an additional reward (as opposed to other options like defining a distribution over actions), since modifying the reward function allows using any standard RL algorithm for policy optimization. We evaluate our approach on Montezuma's Revenge, a challenging game in the Atari domain [Bellemare *et al.*, 2013], demonstrating that it effectively uses linguistic instructions to significantly speed learning, while also being robust to variation in instructions.

2 Overview of the Approach

A Markov Decision Process (MDP) can be defined by the tuple $\langle S, A, T, R, \gamma \rangle$, where S is a set of states, A is a set of actions, $T : S \times A \times S \to [0, 1]$ describes transition probabilities, $R : S \times A \to \mathbb{R}$ is a reward function mapping the current state s_t and current action a_t to real-valued rewards, and $\gamma < 1$ is a discount factor. In this work, we consider an extension of the MDP framework, defined by $\langle S, A, R, T, \gamma, l \rangle$, where $l \in L$ is a language command describing the intended behavior (with L defined as the set of all possible language commands). We denote this language-augmented MDP as MDP+L. Given an MDP(+L), reinforcement learning can be used to learn an optimal policy $\pi^* : S \to A$ that maximizes expected sum of rewards. We use R_{ext} ("extrinsic") to denote the MDP reward function above, to avoid confusion with language-based rewards that we define in Section 4.

In order to find an optimal policy in an MDP+L, we use a two-phase approach:

LanguagE-Action Reward Network (LEARN) In this step, we train a neural network that takes paired (trajectory, language) data from the environment and predicts if the language describes the actions within the trajectory. To train the network, we collect natural language instructions for trajectories in the environment (Section 3).

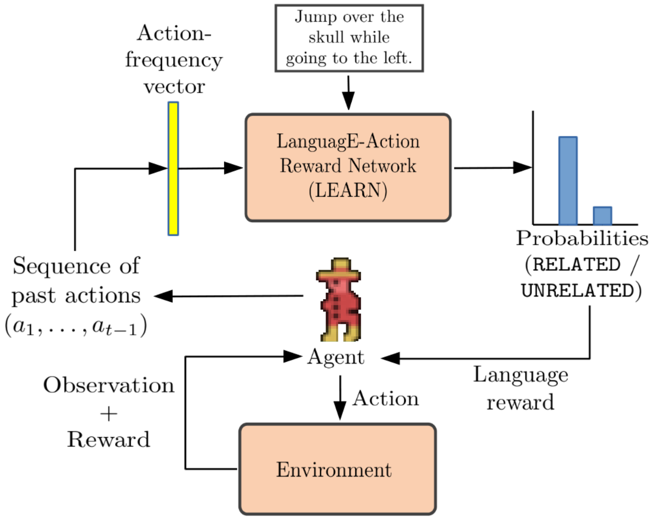


Figure 2: Our framework consists of the standard RL module containing the agent-environment loop, augmented with a LanguagE-Action Reward Network (LEARN) module.

Language-aided RL This step involves using RL to learn a policy for the given MDP+L. Given the trajectory executed by the agent so far and the language instruction, we use LEARN to predict whether the agent is making progress and use that prediction as a shaping reward (Section 4). Note that since we are only modifying the reward function, this step is agnostic to the particular choice of RL algorithm. A schematic diagram of the approach is given in Figure 2.

3 LanguagE-Action Reward Network

3.1 Model

LEARN takes in a trajectory and a language description and predicts whether the language describes the actions in the trajectory. More formally, given a trajectory τ, we create *action-frequency vectors* from it as follows:

1. Sample two distinct timesteps i and j (such that $i < j$) from the set $\{1, \ldots, |\tau|\}$, where $|\tau|$ denotes the number of timesteps in τ. Let $\tau[i : j]$ denote the segment of τ between timesteps i and j.
2. Create an *action-frequency vector* f from the actions in $\tau[i : j]$, where the dimensionality of f is equal to the number of actions in the MDP+L, and the k^{th} component of f is the fraction of timesteps action k appears in $\tau[i : j]$.

Using the above process, we create a dataset of (f, l) pairs from a given set of (τ, l) pairs. Positive examples are created by sampling f from a given trajectory τ and using the language description l associated with τ. Negative examples are created by (1) sampling an action-frequency vector f from a given trajectory τ, but choosing an alternate language description l' sampled uniformly at random from the data excluding l, or (2) creating a random action-frequency vector f' and pairing it with the language description l. These examples are used to train a neural network, as described below. Thus, given a pair (f, l), the network learns to predict whether the action-frequency vector f is related to the language description l or not.

Neural network architecture The action-frequency vector is passed through a sequence of fully-connected layers to get an encoded action vector with dimension D_1. Similarly, the natural language instruction is encoded into a vector with dimension D_2 as described below. The encoded action-frequency vector and language vector are then concatenated, and further passed through another sequence of fully-connected layers, each of dimension D_3, followed by a softmax layer. The final output of the network is a probability distribution over two classes – `RELATED` and `UNRELATED`, corresponding to whether the action-frequency vector f can be explained by the language instruction l.

Language encoder To embed the natural language instruction, we experimented with three models:
(1) **InferSent** : In this model, we used a pretrained sentence embedding model [Conneau *et al.*, 2017], which embeds sentences into a 4096-dimensional vector space. The 4096-dimensional vectors were projected to D_2-dimensional vectors using a fully-connected layer. We train only the projection layer during training, keeping the original sentence embedding model fixed.
(2) **GloVe+RNN** : In this model, we represent the sentence using pretrained 50-dimensional GloVe word embeddings [Pennington *et al.*, 2014], and train a two-layer GRU [Cho *et al.*, 2014] encoder on top of it, while keeping the word embeddings fixed. We used the mean of the output vectors from the top layer as the encoding of the sentence. The hidden state size of the GRUs was set to D_2.
(3) **RNNOnly** : This model is identical to Glove+RNN, except instead of starting with pretrained GloVe vectors, we randomly initialize the word vectors and train both the word embeddings and the two-layer GRU encoder.

These three models trade-off prior domain knowledge with flexibility – InferSent model starts with the knowledge of sentence similarity and is least flexible, GloVe+RNN model starts with word vectors and is more flexible in combining them to generate sentence embeddings, while RNNOnly starts with no linguistic knowledge and is completely flexible while learning word and sentence representations.

Our complete neural network architecture is shown in Figure 3. D_1, D_2 and D_3 were tuned using validation data.

Training procedure We used backpropagation with an Adam optimizer [Kingma and Ba, 2014] to train the above neural network for 50 epochs to minimize cross-entropy loss.

3.2 Data Collection

To collect data for training LEARN, we generate trajectories in the environment, which may or may not be directly relevant for the final task(s). Then, for each trajectory, we get natural language annotations from human annotators, which are in the form of instructions that the agent should follow to go from the initial state of the trajectory to the final state.

In our experiments, we used 20 trajectories from the Atari Grand Challenge dataset [Kurin *et al.*, 2017], which contains hundreds of crowd-sourced trajectories of human gameplays on 5 Atari games, including Montezuma's Revenge. The 20 trajectories we used contain a total of about 183,000 frames. From these trajectories, we extracted 2,708 equally-spaced clips (with overlapping frames), each three-seconds long.

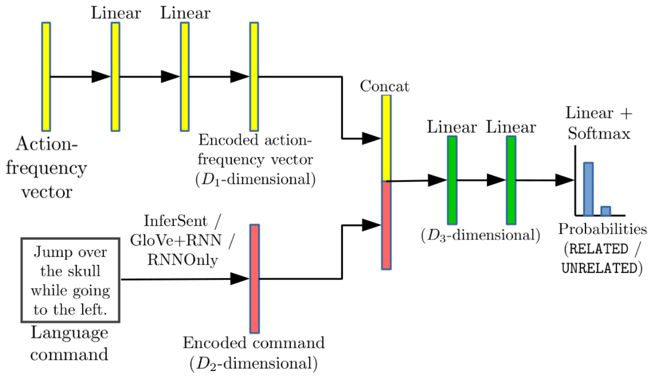

Linear
Linear
Action-frequency vector
Encoded action-frequency vector (D_1-dimensional)
Concat
Linear
Linear
Linear + Softmax
InferSent / GloVe+RNN / RNNOnly
Jump over the skull while going to the left.
Language command
Encoded command (D_2-dimensional)
(D_3-dimensional)
Probabilities (RELATED / UNRELATED)

Figure 3: Neural network architecture for LEARN (Section 3.1)

To obtain language descriptions for these clips, we used Amazon Mechanical Turk. Workers were shown clips from the game and asked to provide corresponding language instructions. Each annotator was asked to provide descriptions for 6 distinct clips, while each clip was annotated by 3 people.

To filter out bad annotations, we manually looked at each set of 6 annotations and discarded the set if any of them were generic statements (e.g. “Good game!”, “Well played.”), or if all the descriptions were very similar to one another (therefore suggesting that they are probably not related to the corresponding clips). After filtering, we obtained a total of 6,870 language descriptions. Note that the resulting dataset may still be quite noisy, since our filtering process doesn't explicitly check if the language instructions are related to the corresponding clips, nor do we correct for any spelling or grammatical errors.

More details about the Amazon Mechanical Turk interface and example descriptions are included in the supplementary material.

4 Using Language-based Rewards in RL

To incorporate language information into RL, we use LEARN's predictions to generate intermediate rewards. Given the sequence of actions $a_1, \ldots, a_{t-1}$ executed by the agent until timestep t and the language instruction l associated with the given MDP+L, we create an action-frequency vector f_t, by setting the k^{th} component of f equal to the fraction of timesteps action k appears in $a_1, \ldots, a_{t-1}$. The resulting action-frequency vector f and the language instruction l are passed to LEARN. Let the output probabilities corresponding to classes `RELATED` and `UNRELATED` be denoted as $p_R(f_t)$ and $p_U(f_t)$. Note that since l is fixed for a given MDP+L, $p_R(f_t)$ and $p_U(f_t)$ are functions of only the current action-frequency vector f_t.

Intuitively, trajectories that contain actions described by the language instruction more often will have higher values of $p_R(f_t)$, compared to other trajectories. For instance, if the language instruction is “Jump over the skull while going to the left”, then trajectories with high frequencies corresponding to the “jump” and “left” actions will be considered more related to the language by LEARN. Therefore, we can use these probabilities to define intermediate language-based rewards. These intermediate rewards will enable the agent to explore more systematically, by choosing relevant actions more often than irrelevant actions.

To map the probabilities to language-based shaping rewards, we define a potential function for the current timestep as $\phi(f_t) = p_R(f_t) - p_U(f_t)$. The intermediate language-based reward is then defined as $R_{lang}(f_t) = \gamma \cdot \phi(f_t) - \phi(f_{t-1})$, where γ is the discount factor for the MDP+L. We show in the supplementary material that a policy that is optimal under the original reward function (R_{ext}) is also optimal under the new reward function ($R_{ext} + R_{lang}$).

5 Experimental Evaluation

To validate the effectiveness of our approach, we conducted experiments on the Atari game Montezuma's Revenge. The game involves controlling an agent to navigate around multiple rooms. There are several types of objects within the rooms – (1) ladders, ropes, doors, etc. that can be used to navigate within a room, (2) enemy objects (such as skulls and crabs) that the agent needs to escape from, (3) keys, daggers, etc. that can be collected. A screenshot from the game is included in Figure 1. We selected this game because the rich set of objects and interactions allows for a wide variety of natural language descriptions.

The first step involved collecting (trajectory, language) pairs in the game as described in Section 3.2. The (trajectory, language) pairs were split into training and validation sets, such that there is no overlap between the frames in the training set and the validation set. In particular, Level 1 of Montezuma's revenge consists of 24 rooms, of which we use 14 for training, and the remaining 10 for validation and testing. The set of objects in both training and validation/test set are the same, but each room has only a subset of these objects arranged in different layouts. We create a training dataset with 160,000 (action-frequency vector, language) pairs from the training set, and a validation dataset with 40,000 pairs from the validation set, which were used to train LEARN.

We define a set of 15 diverse tasks in multiple rooms, each of which requires the agent to go from a fixed start position to a fixed goal position while interacting with some of the objects present in the path.[1] For each task, the agent gets an extrinsic reward of +1 from the environment for reaching the goal, and an extrinsic reward of zero in all other cases.

For each of the tasks, we generate a reference trajectory, and use Amazon Mechanical Turk to obtain 3 descriptions for the trajectory. We use each of these descriptions as language commands in our MDP+L experiments, as described below. Note that we do not use the reference trajectories to aid learning the policy in MDP+L; they are only used to collect language commands to be used in our experiments.

We use Proximal Policy Optimization, a popular on-policy RL algorithm [Schulman *et al.*, 2017]. We train the policy for 500,000 timesteps for all our experiments.

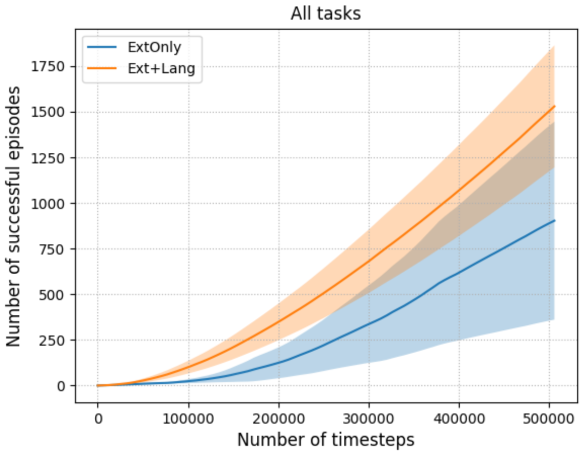


Figure 4: Comparison of different reward functions: The solid lines represent the mean successful episodes averaged over all tasks, and the shaded regions represent 95% confidence intervals.

5.1 How much does language help?

Settings We experiment with 2 different RL setups to evaluate how much using language-based rewards help:
(1) **ExtOnly**: In this setup, we use the original environment reward, without using language-based reward. This is the standard MDP setup, and serves as our baseline.
(2) **Ext+Lang**: In this setup, in addition to the original environment reward that the agent gets on completing the task successfully, we also provide the agent potential-based language reward R_{lang} at each step, as described in Section 4.

Metrics Performance is evaluated using two metrics:
(1) **AUC**: From each policy training run, we plot a graph with the number of timesteps on the x-axis and the number of successful episodes on the y-axis. The area under this curve is a measure of how quickly the agent learns, and is the metric we use to compare two policy training runs.
(2) **Final Policy**: To compare the final learned policy with ExtOnly and Ext+Lang, we perform policy evaluation at the end of 500,000 training steps. For each policy training run, we use the learned policy for an additional 10,000 timesteps without updating it, and record the number of successful episodes.

Hyperparameters For the Ext+Lang setup, we perform validation over the three types of language encoders described in Section 4 (InferSent / GloVe+RNN / RNNOnly). For each type of language encoder, we use the LEARN model with the best accuracy on the validation data. Further, we define the joint reward function as $R_{total} = R_{ext} + \lambda R_{lang}$. The type of language encoder and the hyperparameter λ are selected using validation as described below.

We treat each task as the test task in turn, using the remaining 14 tasks to find the best language encoder and λ. For each setting of the hyperparameters, we run policy training on all the validation tasks and each of the 3 descriptions, and compute AUC for each run. Since AUCs across tasks differ by orders of magnitude (due to varying task difficulties),

[1] Although the tasks (and corresponding descriptions) involve interactions with objects, we observe that just using actions, as we do in our approach, already gives improvements over the baseline, likely because most objects can be interacted with only in one way.

we aggregate the scores across tasks as follows – for each validation task, we compute a rank for each setting of the hyperparameters based on AUC, and then for each setting of the hyperparameters, we compute its average rank across the validation tasks. The setting with the best average rank is used for the test task.

Results At test time, we performed 10 policy learning runs with different initializations for each task and each description. The results, averaged across all tasks and descriptions, are summarized in Figure 4, from which we can conclude that Ext+Lang learns much faster than ExtOnly, demonstrating that using natural language instructions for reward shaping is effective. In particular, the average number of successful episodes for ExtOnly after 500,000 timesteps is 903.12, while Ext+Lang achieves that score only after 358,464 timesteps, which amounts to a 30% speed-up. Alternately, after 500,000 timesteps, Ext+Lang completes 1529.43 episodes on average, compared to 903.12 for ExtOnly, thereby giving a 60% relative improvement.

Statistical Significance Tests For each task, we perform an unpaired t-test between 10 runs of policy training with random initializations using ExtOnly reward function and 30 runs of policy training with random initializations using Ext+Lang reward function (3 descriptions × 10 runs per description), for both metrics.
(1) **AUC**: Of the total 15 tasks, Ext+Lang gives statistically significant improvement in 11 tasks, leads to statistically significant deterioration in 1 task, and makes no statistical difference in the remaining 3 tasks. This agrees with the conclusions from Figure 4, that using language-based reward improves the efficiency of policy training on average.
(2) **Final Policy**: We observe that the number of successful episodes for the final policies is statistically significantly greater for Ext+Lang compared to ExtOnly in 8 out of 15 tasks, while the difference is not significant in the remaining 7 tasks. Further, averaged across all tasks, the number of successful episodes is more than twice with Ext+Lang compared to ExtOnly. These results suggests that using natural language for reward shaping often helps learn a better final policy, and rarely (if ever) results in a worse policy.

5.2 Analysis of Language-based Rewards

In order to analyze if the language-based rewards generated from LEARN actually correlate with language descriptions for the task, we compute the Spearman's rank correlation coefficient between each component of the action-frequency vector and corresponding prediction from LEARN over the 500,000 timesteps of policy training. Correlation coefficients averaged across 10 runs of policy training for some selected tasks are reported in Table 1. Figure 5 shows the policy training curves for these selected tasks.

This analysis supports some interesting observations:
(1) For task 4 with simple descriptions, only the `DOWN` action is positively correlated with language-based reward. All other actions have a strong negative correlation with language-based reward, suggesting that the proposed approach discourages those actions, thereby aiding exploration.
(2) For task 6 with more complex descriptions, LEARN correctly predicts language rewards to be correlated with actions `LEFT` and `DOWN`. For the third description, since the description does not instruct going down, the language reward is negatively correlated with the `DOWN` action. Indeed, we notice in our experiments that we obtain statistically significant improvement in AUC for the first two descriptions, while no statistically significant difference for the third description.
(3) Task 14 represents a failure case. Language rewards predicted by LEARN are not well-correlated with the description, and consequently, using language-based rewards results in statistically significant deterioration in AUC. In general, we observe that groundings produced by LEARN for descriptions involving the word "jump" are noisy. We hypothesize that this is because (i) the `JUMP` action typically appears with other actions like `LEFT` and `RIGHT`, and (ii) humans would typically use similar words to refer to `JUMP`, `JUMP-LEFT` and `JUMP-RIGHT` actions. These factors make it harder for the network to learn correct associations.

Note that LEARN does not see action names used in Table 1 (`NO-OP`, `JUMP`, etc.); instead, actions are represented as ordinals from 0 through 17. Thus, we see that our approach successfully learns to ground action names to actions in the environment.[2]

6 Related Work

Prior work on combining RL and natural language can be divided into two classes. The first class uses reinforcement learning to solve NLP tasks, such as summarization [Paulus *et al.*, 2017], question-answering [Xiong *et al.*, 2017] and dialog generation [Li *et al.*, 2016]. The second class, in which our approach lies, uses natural language to aid RL.

Regarding methods that use NLP to help RL, some recent approaches map natural language to a reward function. [Williams *et al.*, 2017] and [Arumugam *et al.*, 2017] map language to a reward function in an object-oriented MDP framework. However, these approaches use a predefined set of objects, object properties and spatial relations, and/or use simple language-based features, which is difficult to scale to more complex environments and instructions. Our approach, on the other hand, learns to ground natural language concepts to actions directly from data.

[Misra *et al.*, 2017] use natural language to describe the goal, which is combined with the state information to learn a policy in contextual bandit setting. However, they use distance from the goal and from reference trajectories for reward shaping. [Kuhlmann *et al.*, 2004] map natural language to a set of rules which are then used to increase or decrease the probability of choosing an action during reinforcement learning. Extending this to complex environments would require engineering how each rule affects the probabilities of different actions. Our approach, on the other hand, uses the natural language instruction itself for reward shaping, directly generating rewards from language, thereby reducing human effort.

[Branavan *et al.*, 2012b] extract features from natural language instructions, and incorporate them into the action-value

[2] While there are a total of 18 actions, we only report the most common 8 actions in Table 1 for brevity. The omitted 10 actions jointly constitute less that 1% of the actions in the training data.

Task Id	Description	Correlation coefficients of different actions							
		NO-OP	JUMP	UP	RIGHT	LEFT	DOWN	JUMP-RIGHT	JUMP-LEFT
4	climb down the ladder	-0.60	-0.58	-0.59	-0.61	-0.55	0.07	-0.57	-0.56
	go down the ladder to the bottom	-0.58	-0.58	-0.58	-0.60	-0.53	0.09	-0.59	-0.60
	move on spider and down on the lader	-0.58	-0.54	-0.59	-0.60	-0.49	0.10	-0.58	-0.56
6	go to the left and go under skulls and then down the ladder	-0.37	-0.40	-0.49	-0.43	0.33	0.16	-0.46	-0.01
	go to the left and then go down the ladder	-0.24	-0.26	-0.35	-0.31	0.28	0.36	-0.34	-0.04
	move to the left and go under the skulls	-0.16	-0.25	-0.60	-0.48	0.27	-0.63	-0.52	-0.40
14	Jump once then down	0.00	0.07	-0.15	-0.13	0.51	0.50	0.09	0.52
	go down the rope and to the bottom	-0.03	0.10	-0.16	0.56	0.54	0.33	0.28	0.01
	jump once and climb down the stick	0.11	0.11	0.06	0.04	0.14	0.40	0.25	0.11

Table 1: Analysis of language-based rewards

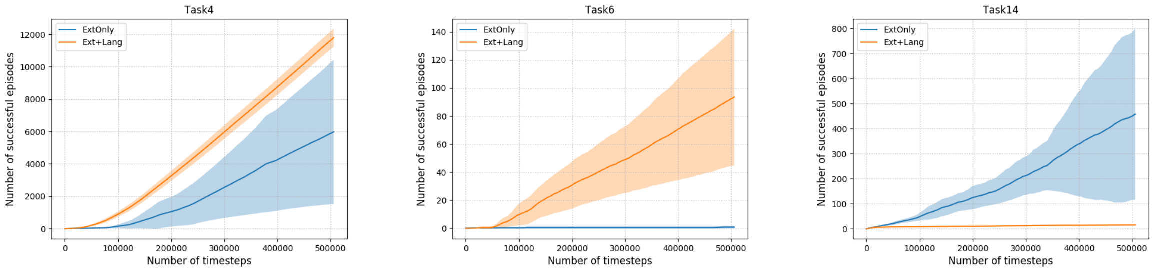


Figure 5: Comparisons of different reward functions for selected tasks

function. More recently, [Bahdanau *et al.*, 2018] proposed an adversarial learning framework wherein a discriminator distinguishes between a fixed set of good (instruction, state) pairs and (instruction, state) pairs generated by the current policy, and this is used as a reward function to simultaneously improve the policy. A key difference between these approaches and our approach is that they learn linguistic features jointly during reinforcement learning, while we learn to map language to a reward function offline, which could be beneficial if interaction with the environment is expensive. However, our approach requires pairs of trajectories and natural language instructions for offline training.

[Branavan *et al.*, 2012a] and [Kaplan *et al.*, 2017] use natural language to do high-level planning. These approaches are orthogonal to our work, in that these approaches can be used to generate subgoals at a high-level, whereas our approach can be used to make exploration faster at a lower-level.

Finally, our model is related to that in [Wang *et al.*, 2018], which also uses intermediate language-based rewards in RL. However, their goal is to use RL to improve natural language instruction-following, while our focus is on the reverse problem of using instructions to improve RL performance.

7 Conclusions and Future Work

We propose LanguagE Action Reward Network (LEARN), a framework trained on paired (trajectory, language) data in an environment to predict if the actions in a trajectory match the language description. The outputs of the network are used to generate intermediate rewards for reinforcement learning. We show in our experiments that these language-based rewards can be used to train faster and learn a better policy for sparse reward settings. Further, since the modality by which information is given to the agent is natural language, this approach can potentially be used even by non-experts to specify tasks to RL agents.

While our approach achieves promising improvements over the baseline, there are several possible extensions of the approach:

1) **Temporal ordering:** Our approach aggregates the sequence of past actions into an action-frequency vector, thereby losing temporal information. Therefore a possible extension is to look at the complete action sequences.

2) **State-based rewards:** Currently, the language-based reward is a function of only the past actions. As such, the model cannot utilize natural language descriptions that refer to objects in the state (e.g. “Go towards the *ladder*”, “avoid the *skulls*”.) Modelling the language-based reward as a function of both the past states and actions should allow the agent to benefit from such language descriptions.

3) **Multi-step instructions:** The current approach only handles a single instruction. One way to handle multiple instructions is to have another module (trained / heuristic-based) to predict if a language instruction has been completed or not. This could then be used in conjunction with our current approach, where the agent starts following the first instruction, and transitions to the next one when this new module predicts that the current instruction has been completed.

Acknowledgements

This work was partially supported by an NSF NRI grant (IIS-1637736) and an Amazon Research Award.

References

[Arumugam *et al.*, 2017] Dilip Arumugam, Siddharth Karamcheti, Nakul Gopalan, Lawson LS Wong, and Stefanie Tellex. Accurately and efficiently interpreting human-robot instructions of varying granularities. *arXiv preprint arXiv:1704.06616*, 2017.

[Bahdanau *et al.*, 2018] Dzmitry Bahdanau, Felix Hill, Jan Leike, Edward Hughes, Pushmeet Kohli, and Edward Grefenstette. Learning to follow language instructions with adversarial reward induction. *arXiv preprint arXiv:1806.01946*, 2018.

[Bellemare *et al.*, 2013] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[Branavan *et al.*, 2012a] SRK Branavan, Nate Kushman, Tao Lei, and Regina Barzilay. Learning high-level planning from text. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pages 126–135. Association for Computational Linguistics, 2012.

[Branavan *et al.*, 2012b] SRK Branavan, David Silver, and Regina Barzilay. Learning to win by reading manuals in a monte-carlo framework. *Journal of Artificial Intelligence Research*, 43:661–704, 2012.

[Cho *et al.*, 2014] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[Conneau *et al.*, 2017] Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. Supervised learning of universal sentence representations from natural language inference data. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 670–680, Copenhagen, Denmark, September 2017. Association for Computational Linguistics.

[Harnad, 1990] Stevan Harnad. The symbol grounding problem. *Physica D: Nonlinear Phenomena*, 42(1-3):335–346, 1990.

[Kaplan *et al.*, 2017] Russell Kaplan, Christopher Sauer, and Alexander Sosa. Beating atari with natural language guided reinforcement learning. *arXiv preprint arXiv:1704.05539*, 2017.

[Kingma and Ba, 2014] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[Kostrikov, 2018] Ilya Kostrikov. Pytorch implementations of reinforcement learning algorithms. https://github.com/ikostrikov/pytorch-a2c-ppo-acktr, 2018.

[Kuhlmann *et al.*, 2004] Gregory Kuhlmann, Peter Stone, Raymond Mooney, and Jude Shavlik. Guiding a reinforcement learner with natural language advice: Initial results in robocup soccer. In *The AAAI-2004 workshop on supervisory control of learning and adaptive systems*. San Jose, CA, 2004.

[Kurin *et al.*, 2017] Vitaly Kurin, Sebastian Nowozin, Katja Hofmann, Lucas Beyer, and Bastian Leibe. The atari grand challenge dataset. *arXiv preprint arXiv:1705.10998*, 2017.

[Li *et al.*, 2016] Jiwei Li, Will Monroe, Alan Ritter, Michel Galley, Jianfeng Gao, and Dan Jurafsky. Deep reinforcement learning for dialogue generation. *arXiv preprint arXiv:1606.01541*, 2016.

[Misra *et al.*, 2017] Dipendra Misra, John Langford, and Yoav Artzi. Mapping instructions and visual observations to actions with reinforcement learning. *arXiv preprint arXiv:1704.08795*, 2017.

[Ng *et al.*, 1999] Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, pages 278–287, 1999.

[Paulus *et al.*, 2017] Romain Paulus, Caiming Xiong, and Richard Socher. A deep reinforced model for abstractive summarization. *arXiv preprint arXiv:1705.04304*, 2017.

[Pennington *et al.*, 2014] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[Schulman *et al.*, 2017] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[Večerík *et al.*, 2017] Matej Večerík, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothörl, Thomas Lampe, and Martin Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817*, 2017.

[Wang *et al.*, 2018] Xin Wang, Qiuyuan Huang, Asli Celikyilmaz, Jianfeng Gao, Dinghan Shen, Yuan-Fang Wang, William Yang Wang, and Lei Zhang. Reinforced cross-modal matching and self-supervised imitation learning for vision-language navigation. *arXiv preprint arXiv:1811.10092*, 2018.

[Williams *et al.*, 2017] Edward C Williams, Mina Rhee, Nakul Gopalan, and Stefanie Tellex. Learning to parse natural language to grounded reward functions with weak supervision. In *AAAI Fall Symposium on Natural Communication for Human-Robot Collaboration*, 2017.

[Xiong *et al.*, 2017] Caiming Xiong, Victor Zhong, and Richard Socher. Dcn+: Mixed objective and deep residual coattention for question answering. *arXiv preprint arXiv:1711.00106*, 2017.

A Example Annotations

Table 2 shows 20 randomly selected annotations collected using Amazon Mechanical Turk (after the filtering process described in Section 3.2). Note that the annotations have a significant amount of variation, both in terms of length and vocabulary. Further, several descriptions (1) contain spelling errors (e.g. "climbling" in annotation 6 and "dwon" in annotation 7), (2) are ill-formed (e.g. annotation 2) or (3) are not very informative (e.g. annotations 1 and 7). We do not filter out or correct these annotations, as the process requires significant manual effort. Thus, our method is able to extract useful information from these annotations even in the presence of noise.

1.	wait
2.	using the ladder on standing
3.	going slow and climb down the ladder
4.	move down the ladder and walk left
5.	go left watch the trap and move on
6.	climbling down the ladder
7.	ladder dwon and running this away
8.	stay in place on the ladder.
9.	go down the ladder
10.	go right and climb up the ladder
11.	just jump and little move to right side
12.	run all the way to the left.
13.	go left jumping once
14.	go left
15.	move right and jump over green creature then go down the ladder
16.	hop over to the middle ledge
17.	wait for the two skulls and dodge them in the middle
18.	walk to the left and then jump down
19.	jump to collected gold coin and little move
20.	wait for the platform to materialize then walk and leap to your right to collect the coins.

Table 2: Examples of descriptions collected using Amazon Mechanical Turk

B Policy Invariance

In this section, we show that using action-frequency vectors for reward shaping does not change the optimal policy.

Theorem. *Let $M = \langle S, A, T, R, \gamma \rangle$ be a given MDP, and $R_{lang}(f_t) = \gamma \cdot \phi(f_t) - \phi(f_{t-1})$ be a shaping reward function, where f_t is the action-frequency vector corresponding to actions $a_1, \ldots, a_t$ as defined in Section 3.1, and ϕ be a potential function. Then, an optimal policy in M is also an optimal policy in the MDP $M' = \langle S, A, T, R + F, \gamma \rangle$.*

Proof. Define an MDP $\widehat{M} = \langle \widehat{S}, \widehat{A}, \widehat{T}, \widehat{R}, \gamma \rangle$, such that

- For all $s \in S$ and $g \in \mathbb{Z}_+^{|A|}$, $(s, g) \in \widehat{S}$.
  (g is the vector of counts of each action.)
- $\widehat{A} = A$.
- $\widehat{R}((s,g), a, (s', g')) = R(s, a, s')\mathbb{1}[g, a, g'\text{consistent}]$
  (Consistent refers to whether g' is obtained from g on taking action a.)
- $\widehat{T}((s,g), a, (s', g')) = T(s, a, s')\mathbb{1}[g, a, g'\text{consistent}]$

Let Q^*_M be the optimal Q-function for the original MDP M. Define

$$\widehat{Q}_{M'}((s,g), a) = Q^*_M(s, a)$$

Now,

$$\begin{aligned}
&\mathbb{E}_{(s',g')\sim\widehat{T}}[\widehat{R}((s,g), a, (s',g')) + \gamma \max_{a'} \widehat{Q}_{M'}((s',g'), a')] \\
&= \mathbb{E}_{(s',g')\sim\widehat{T}}[R(s,a,s')\mathbb{1}[f, a, g'\text{consistent}] + \gamma \max_{a'} Q^*_M(s', a')] \\
&= \mathbb{E}_{s'\sim T}[R(s,a,s') + \gamma \max_{a'} Q^*_M(s', a')] \\
&= Q^*_M(s, a) \\
&= \widehat{Q}_{M'}((s,g), a)
\end{aligned} \tag{1}$$

The second step involves expanding out the expectation w.r.t. $\widehat{T}$, removing the inconsistent terms, since they get multiplied by zero, and converting back to expectation w.r.t. T.
Thus, $\widehat{Q}_{M'}$ satisfies the Bellman optimality equation for M'.

Next, let π^*_M be an optimal policy for M. Then,

$$\pi^*_M(s) \in \arg\max_a Q^*_M(s, a)$$

Defining $\widehat{\pi}_{M'}((s,g)) = \pi^*_M(s)$, we get

$$\begin{aligned}
\pi_{M'}((s,g)) &= \pi^*_M(s) \\
&\in \arg\max_a Q^*_M(s, a) \\
&= \arg\max_a \widehat{Q}_{M'}((s,f), a)
\end{aligned} \tag{2}$$

Using equations 1 and 2, we can conclude that $\widehat{\pi}_{M'}((s,g))$ is optimal in M'.

Note that M' could admit other optimal policies as well, which could potentially also depend on g.

Since the states in M' contain the history of action counts, our proposed potential-based shaping reward can now be defined as a function of only the state in M'. From [Ng *et al.*, 1999], these shaping rewards do not change the optimal policy. □

C Sensitivity Analysis

To better understand the relation between the LEARN module and RL, we added varying amounts of noise to the output of LEARN. Specifically, Gaussian noise $\mathcal{N}(0, \sigma)$ was added to the potential function as described in Section 4, where σ was varied from 0.01 to 1.0. The results for Task 8 are shown in Figure 6, from which we can see that the language-based rewards improve over the baseline even with significant amounts of noise. This suggests that the predictions from the LEARN module are fairly robust.

D Amazon Mechanical Turk interface

Figure 7 shows the interface used on Amazon Mechanical Turk for data collection.

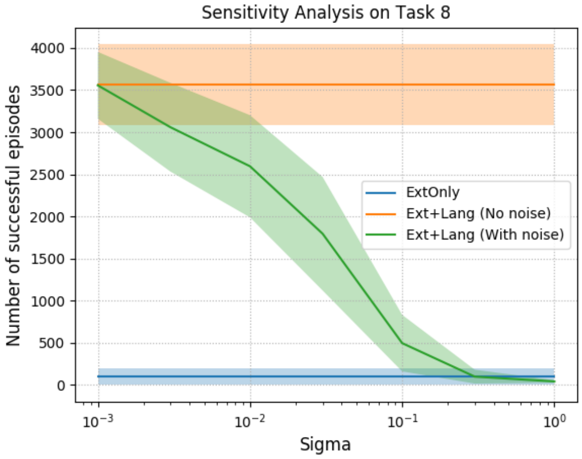
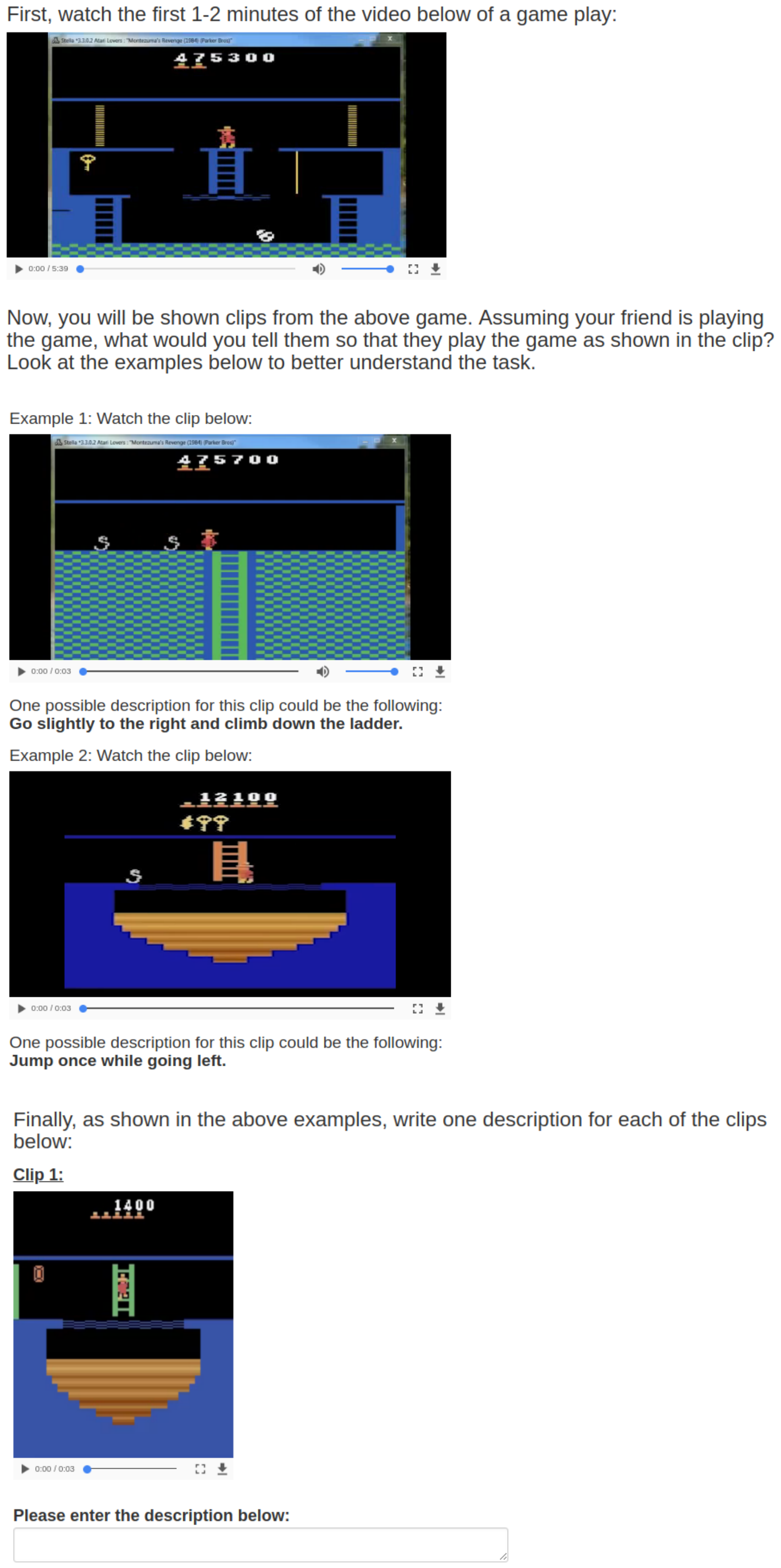


Figure 6: Effect of adding noise to the predictions of LEARN: The solid lines represent the mean successful episodes averaged over all tasks, and the shaded regions represent 95% confidence intervals.



First, watch the first 1-2 minutes of the video below of a game play:

Now, you will be shown clips from the above game. Assuming your friend is playing the game, what would you tell them so that they play the game as shown in the clip? Look at the examples below to better understand the task.

Example 1: Watch the clip below:

One possible description for this clip could be the following:
Go slightly to the right and climb down the ladder.

Example 2: Watch the clip below:

One possible description for this clip could be the following:
Jump once while going left.

Finally, as shown in the above examples, write one description for each of the clips below:

Clip 1:

Please enter the description below:

Figure 7: Sample Mechanical Turk HIT for collecting natural language descriptions.