Improving Reward Shaping via Language Instruction

Yutong Yan
Student
McGill University
Mila-Quebec AI Institute

Irene Zhang
Student
McGill University
Mila-Quebec AI Institute

Siva Reddy
Mentor
McGill University
Mila-Quebec AI Institute

Abstract

Recent literature have started exploring problems in language-assisted reinforcement learning (RL). This is a setting where language itself is not necessary, but incorporating language can be advantageous for solving the task. In this work, We explore one specific case of language-assisted RL, where we address the challenges of solving tasks with sparse reward. We do so by using reward shaping through leveraging natural language instructions. Our project extends a recent work, in which a framework called LanguagE-Action Reward Network (LEARN) is proposed. LEARN maps free-form natural language instructions to intermediate rewards based on actions taken by the agent. The work has shown that language-based rewards can lead to performance advancement compared to learning without language through extensive experiments. However, one draw back of the method is that the reward shaping is only dependent on the actions for simplicity. In this work, we would like to extend the approach by learning a state representation using a VAE, and conduct reward shaping based on both actions and the state representation. Our empirical results show that the VAE is not able to learn representations that can significantly improve baselines, but other methods of representation learning for states should be investigated in future work.

1 Introduction

1.1 Motivation

In reinforcement learning (RL), the task is represented by the reward function. The optimal policy is determined by the reward function and the model of the domain. In the past decade, RL approaches have shown great success in solving many complex sequential decision-making problems (Mnih et al., 2013; Silver et al., 2017). However, existing solutions often suffer from poor performance and sample inefficiency when the reward function gives very sparse signals. This is known as the "sparse reward" problem in RL. A common solution is through reward shaping, which is the practice of modifying the reward function by supplying additional rewards (Ng et al., 1999).

Figure 1: The Atari game Montezuma's Revenge is a classic task which shows the difficulty for solving sparse reward problems. Through out the game, the agent must perform long sequences of actions before receiving a reward at checkpoints.

1.2 Background

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, $T : S \times A \times A \to [0, 1]$ 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.

Recently, (Goyal et al., 2019) proposes a framework that matches free-form natural language instructions to intermediate rewards based on actions taken by the agent (shown in fig. 6). The LanguagE-Action Reward Network (LEARN) framework consists of two modules:

1) A neural network that takes the agent's action-frequency sequence f and natural language instruction l as input, and predicts whether the sequence

is related to the instruction or not with two probabilities, $P_R(f)$ and $P_U(f)$, corresponding to the classes RELATED and UNRELATED respectively. The action-frequency vector, with dimensionality equal to the number of actions in the environment, is created by first sampling two time steps i and j ($i < j$), and the k-th element of the vector is the fraction of time steps action k appears between i and j. The language instruction is either embedded with pretrained embeddings or recurrent neural networks (RNN).

2) The reward shaping step is essentially adding an additional term, the difference between RELATED and UNRELATED probabilities outputted by the neural network, to the reward: The potential function is defined as $\phi(f_t) = P_R(f) - P_U(f)$. The intermediate reward is then defined as $R_{lang}(f_t) = \gamma \cdot \phi(f_t) - \phi(f_{t-1})$, where γ is the discount factor. The language-based reward shaping term R_{lang} is then added to the original reward function to solve for the optimal policy.

In this work, the standard MDP is modified as language-augmented MDP, denoted as MDP+L, shown in Figure 6. MDP+L is defined by the tuple $\langle S, A, T, R, \gamma, l \rangle$, where $l \in L$ is a language command describing the intended behavior. In this work, L is defined as the set of all possible language commands in the environment. Our goal is to learn the optimal policy $\pi^* : S \times A$ that maximizes the expected sum of rewards. We use R_{ext} (extrinsic reward) to denote standard reward function defined above, in order to distinguish the intrinsic reward R_{lang}.

1.3 Contribution

We wish to extend LEARN by adding aState-based rewards: In LEARN, the language-based reward is a function of only the past actions. However, (Wang et al., 2019) has shown that memorizing the past can enable the recognition of the current status and thus understanding which words or sub-instructions to focus on next. Therefore, modelling the language-based reward as a function of both the past states and actions should allow the agent to benefit from the language descriptions that refer to objects in the state.

2 Literature Review

LEARN is the first work to use natural language instruction to assist learning. We are not able to relate similar works with the same focus. However,

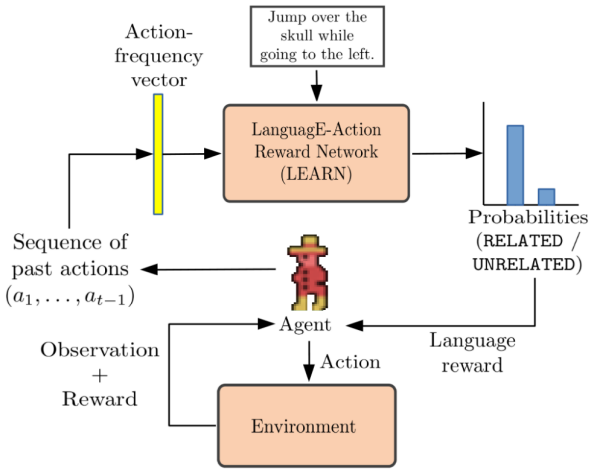

Figure 2: Standard RL module augmented with a LEARN module.

there have been several related works leveraging natural language instructions in RL tasks, some examples of these work includes the following:

2.1 Learning instruction conditioned policy in RL

Instead of learning a language-conditional reward that has been done in LEARN, (Bahdanau et al., 2018) proposes the method, Adversarial Goal-Induced Learning from Examples (AGILE) that learns a language-conditional *policy* π_θ. AGILE is designed for tasks where the reward function needs to be learned or when the reward is sparse. In AGILE, the agent is given instructions describing the goal state which the agent should reach. The AGILE framework can be decomposed into two modules, a discriminator and a generator, as seen in (Goodfellow et al., 2014) and (Ho and Ermon, 2016):

1) The Reward Discriminator $\mathcal{D}_\phi$: The discriminator is given pairs of natural language instructions c and and state s and tries to discriminate whether the a given state is a goal state for the instruction by minimizing the loss function. (c, s) are negative example pairs drawn randomly from a buffer, while (c_i, s_i) are positive examples draw from dataset $\mathcal{D}$. The loss function is defined as $L_D(\phi) = -\mathbb{E}_{(c,g)\sim\mathcal{B}} \log\left(1 - D_\phi(c, s)\right) - \mathbb{E}_{(c_i,g_i)\sim\mathcal{D}} \log D_\phi\left(c_i, g_i\right)$. The discriminator provides a meaningful reward function for training π_θ:

$$\hat{r}_t = \begin{cases} 1 & \text{if } D_\phi(c, s_t) > 0.5. \\ 0 & \text{o.w.} \end{cases}$$

2) The Policy Generator R: Using the reward function $\hat{r}$ provided by the discriminator, we can then train a policy π_θ normally like any other reinforcement learning problem, where we find θ by maximizing the expected total reward: $R_\pi(\theta) = \mathbb{E}_{(c,s_{1:\infty})} \sum_{t=1}^{\infty} \gamma^{t-1} \hat{r}_t + \alpha H(\pi_\theta)$.

Here, H is the entropy of the policy for regularization purposes, and α is a hyper-parameter.

Overall, AGILE uses an adversarial framework, composed of a reward model that recognizes goal states from language instructions, and a policy that learns what to do to get to a goal state.

2.2 Improving instruction-following using RL

In Wang et al. 2019, intermediate language-based rewards are also used in RL. The focus of the work is to learn a better and more generalizable using an intrinsic reward function. Compared to LEARN, this work uses RL to improve natural language instruction-following, while the goal of LEARN is to use instructions to facilate RL training.

In particular, a matching critic provides an intrinsic reward to encourage the matching between instructions and trajectories. The goal of the intrinsic reward signal is to encourage the RL agent to better understand the language input and penalize the trajectories that do not match the instructions. Being trained with the extrinsic reward from the environment and intrinsic reward from the matching critic, the agent learns to ground the natural language on the trajectory. The intrinsic reward is constructed as the probability of the language instruction, given the trajectory executed. In this work, the matching critic is pretrained with human demonstrations (the ground-truth instruction-trajectory pairs) via supervised learning. Interestingly, the total reward is composed of the extrinsic reward and a weighing intrinsic reward, with a weighing hyperparameter. It is worth our attention that if we should weigh our intrinsic reward.

3 Models

3.1 Baselines

We conducted experiments with two baselines. The first one is simply the standard RL agent with no reward shaping (e.g. the agents receives reward only from the environment). This is to test the hypothesis whether reward shaping improves RL agent training in sparse reward problems.

The second baseline used is the standard RL agent augmented with the LEARN module from

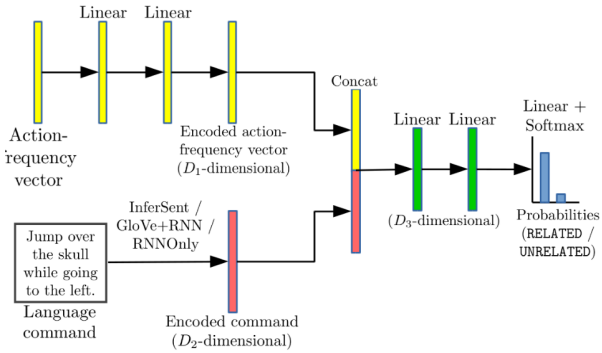

Figure 3: LEARN module with action representation input only

Goyal et al.'s original paper (refer to figure 3). The LEARN module takes in two inputs:

Action frequency vector Representation of the sequence of actions described in section 1.2.

Embedded natural language instruction The instructions were embedded using either of the three following encoders:

- **InferSent** A pretrained sentence embedding model (Conneau et al., 2017) with one fully connected layer. Only the connected layer is trained during training.
- **GloVe + RNN**: A pretrained word embedding (Pennington et al., 2014) with two-layer GRU encoder. The mean of the top encoder is used as the sentence embedding. Only the GRU encoder is trained during training.
- **RNN**: Randomly initialized word embeddings with two-layer GRU. Everything is trained during training.

The LEARN module then outputs a confidence score for the relatedness of the natural language instruction and action sequence, which is used as the reward shaping term in addition to the external reward the agent receives from the environment.

The LEARN baseline with only action representation input is used to test the hypothesis whether state-action dependent reward shaping outperforms action only dependent reward shaping.

3.2 LEARN with State Information

Inspired by (Ha and Schmidhuber, 2018), in which they learn a model via training in an unsupervised manner to learn a compressed spatial and temporal representation of the environment. The original

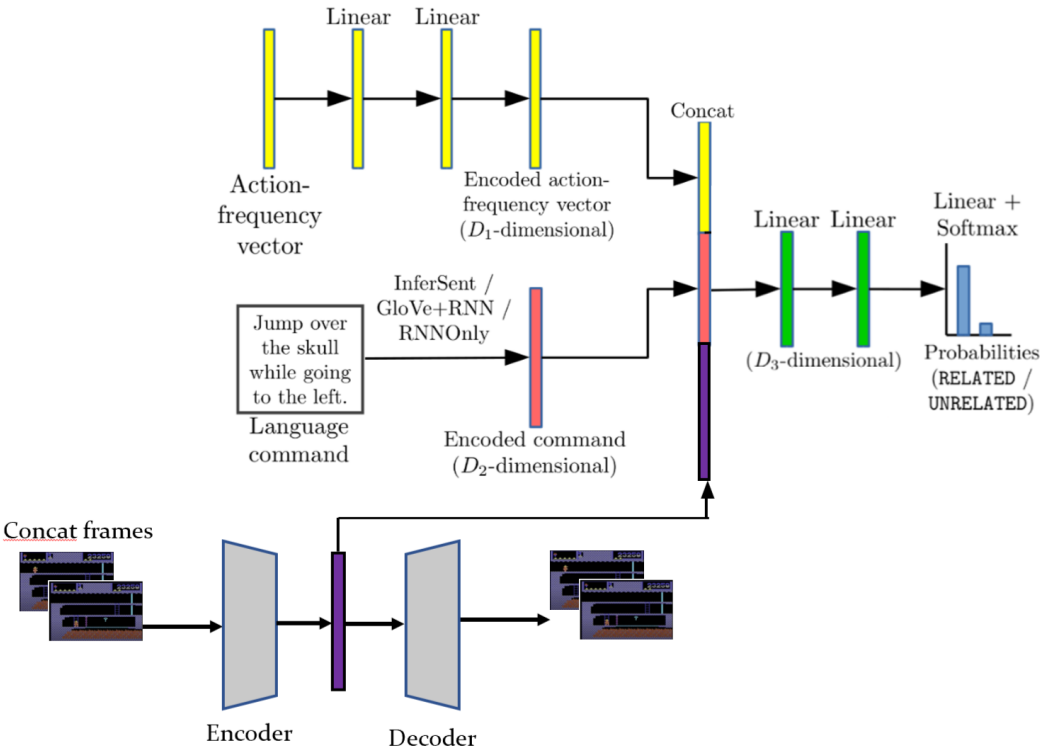

Figure 4: Extended LEARN module with state-action representation

work of LEARN concatenates the action frequency vector and the encoded language command vector. We claim that this information to some extent is not sufficient for the agent to learn the meaning of the language command, without the context of the states. In this work, with the hypothesis that adding extra state information can aid the agent to learn better in an environment where the rewards are sparse, we propose to use a variational autoencoders (VAEs) model (Kingma and Welling, 2013) to learn a state representation and add the latent state representation to the original LEARN module.

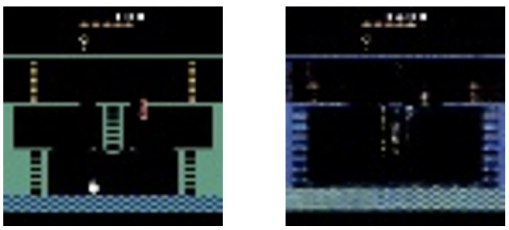

Figure 5: An output example of the learned VAE model

We reuse the same game frame dataset used to collect the annotations for LEARN to learn the VAE model. However, the game frames are repetitive in the dataset, in total of 180, 000. Thereafter, we decide to sample 5 frames uniformly for each annotation so that we guarantee that we are able to generate meaningful state representation for each annotation sample in the collected dataset. Note that each annotation is associated with 150 frames and for efficiency, we decide not to include all frames for one annotation. After we fully pretrain our VAE model, we proceed to the new LEARN module with the extra state information.

Using our learned VAE's encoder, we could take input any game state, and output the latent state representation vector, which we will use to train the LEARN model, in addition to the encoded language command vector and action-frequency vector. In the dataset, each sample is composed of the set of actions taken and the language command. Thereafter, we backtrack using the language command, in the annotation file, to find the corresponding set of games frames. Since the language command is associated with a truncated trajectory, instead of just a game state, we decide to sample 5 game frames from all related game frames, for a sample. For example, with one language command, "go left jumping once", the agent needs to perform multiple actions through many states, like "go left" and "jump". This results in 5 latent state

vectors and we concatenate the vectors into one vector, which is our state representation vector. An example of the decoded game state output by the model is shown in Figure 5.

The fully trained VAE module with VAE is used in the RL framework the same as the original VAE module. The LEARN module takes input a set of actions taken and the encoded language command, additionaly, concatenated latent state vectors of past 5 game frames, since we train our LEARN with 5 game frames. Then, the LEARN module outputs the logits for the related and the unrelated class.

3.3 Dataset

The work in which LEARN is proposed provides complete dataset for training. The trajectories in the environment are first generated, which may or may not be directly relevant for the final task(s). Then, for each trajectory, natural language annotations are marked by human annotators, which act as instructions for the agent to follow from the initial state of the trajectory to the final state. 20 trajectories are selected from the Atari Grand Challenge dataset (Kurin et al., 2017), which contains hundreds of crowd-sourced trajectories of human game plays on 5 Atari games, including Montezuma's Revenge. The 20 trajectories contain a total of about 183,000 frames. From these trajectories, 2,708 equally-spaced clips (with overlapping frames) are extracted, each three-seconds long. Annotators were shown clips from the game and asked to provide corresponding language instructions. Bad annotations and similar annotations were discarded. There are a total number of 6,870 language descriptions. The annotations have a significant amount of variation, both in terms of length and vocabulary, and they are not filtered out or corrected.

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. The training dataset is composed of 160,000 (action-frequency vector, language) pairs and a validation dataset is composed of 40,000 pairs from the validation set. For each task in the game, 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.

3.4 Evaluation Metric

Performance is evaluated using two metrics:

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.

Figure 6: Examples of descriptions collected.

1) AUC: The area under this curve during training, where the number of timesteps is on x-axis and the number of successful episodes is on the y-axis. AUC is a measure of how quickly the agent learns, and is the metric we use to compare two policy training runs.

2) Final Policy: For the last 10,000 time steps, we do not update the policy and record the number of successful episodes (i.e. number of goals reached) using the learned policy.

4 Experiments Details

Defining the Tasks To empirically evaluate the performance of our LEARN model extended with state information against the baselines, we conduct experiments on the Atari game Montezuma's Revenge, a classic example of a sparse reward task. Goyal et al. defined 15 tasks in Montezuma's Revenge, and each task requires the agent to go from a fixed start position to a fixed end position by interacting with objects present along the path. The agent gets an extrinsic reward of +1 from the environment for reaching the goal state, and 0 in all other cases. In this work, we ran empirical experiments on 3 out of 15 tasks - task 4, 6, and 14, due to computation constraints.

To train the RL agent, we used Proximal Policy Optimization (Schulman et al., 2017) for 500,000

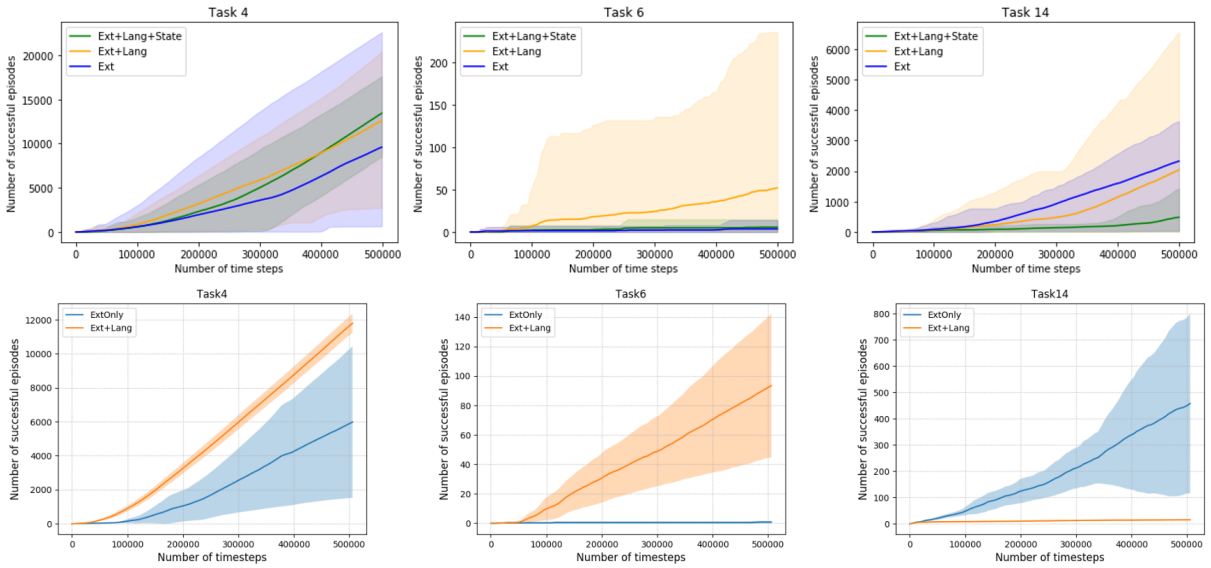

Figure 7: Comparison of AUC for different model and baselines. Top: Our Results. Bottom: Baselines from the original LEARN paper.

Model	Task 4	Task 6	Task 14	Mean
No Reward Shaping (Original Results)	N \A	N \A	N \A	903
No Reward Shaping	11761 76.6%	6382 0.0644%	15434 15.2%	11192 30.7%
LEARN (Original Results)	N \A	N \A	N \A	1529
LEARN	15379 81.1%	12797 0.445 %	16069 16.9 %	14748 32.8%
LEARN with State Information	16443 84.2 %	5132 0.0866%	12377 5.30%	11317 29.9%

Table 1: Experiment results on the 3 tasks, including the average number of successful episodes completed (left) and percentage of successful completion of task (right) during the 10,000 step test run.

time steps for all our experiments. For each task, we trained 9 different policies: Each task was trained with 3 different initialization and 3 different set of language descriptions. We evaluate the performance of the model on a task by averaging the performance over all 9 runs.

Hyper-Parameters For the LEARN module, we select the best encoder (InferSent / GloVe+RNN / RNN) based on the validation accuracy (In our case, InferSent was selected). For the RL module, we tune for the hyper-parameter $\lambda = 1.0, 0.1, 0.01$ which defines the reward function $R_{total} = R_{external} + \lambda R_{language}$. We hyper-tune the λ for each of the task by treating all other task as the validation task. The area under the curve (AUC) is calculated for the validation tasks, and the hyper-parameter with the highest AUC is selected for the task.

5 Results and Discussion

From table 1 and figure 7, we see a comparison of the baselines from the original LEARN paper and our reproduced results. The results agree in terms of ordering, but the magnitude of the results differs significantly. The difference is due to three reasons.

- In our experiments, the mean performance is computed from 3 tasks only rather than all 15 due to computation constraints.
- Our experiment used different initialization seeds, as the seeds used by the original paper were not provided. In addition, the original paper ran 10 seeds for each experiment, while we only used 3 seeds due to computation constraints.
- Our experiment's hyper-parameters tuned were likely different from the original paper, as the authors did not mention what values

the hyper-parameters were tuned with (i.e. we selected λ from $0.01, 0.1, 1.0$, but the original paper did not mention what values they have tried).

Observing the results for the extended LEARN model with state information, we see that the learned state representation does not significantly improves the baselines. This does not necessarily suggest that state representation is not useful for reward shaping, but rather using the latent representation learnt from a VAE is not the correct way to incorporate the representation. Alternative representation learning methods such as using mutual information maximization (Anand et al., 2019) should be explored in future work.

6 Conclusion

In this work, we introduced an extended LEARN model that incorporates both state and action information for reward shaping in the sparse reward setting. The empirical results show that using a VAE is not able to learn state representation that can significantly improve baselines. However, other methods of state representation learning should be investigated in future work. For code, see GitHub repository.

7 Contribution

Yutong Yan processed the dataset to label game states images with instructions, implemented VAE model and ran experiments with LEARN module with state information.

Irene Zhang modified code from original LEARN paper for baseline experiments, ran baseline experiments, and visualized result for baseline and new model.

References

Ankesh Anand, Evan Racah, Sherjil Ozair, Yoshua Bengio, Marc-Alexandre Côté, and R Devon Hjelm. 2019. Unsupervised state representation learning in atari. In *Advances in Neural Information Processing Systems*, pages 8766–8779.

Dzmitry Bahdanau, Felix Hill, Jan Leike, Edward Hughes, Arian Hosseini, Pushmeet Kohli, and Edward Grefenstette. 2018. Learning to understand goal specifications by modelling reward. *arXiv preprint arXiv:1806.01946*.

Alexis Conneau, Douwe Kiela, Holger Schwenk, Loic Barrault, and Antoine Bordes. 2017. Supervised learning of universal sentence representations from natural language inference data. *arXiv preprint arXiv:1705.02364*.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

Prasoon Goyal, Scott Niekum, and Raymond J Mooney. 2019. Using natural language for reward shaping in reinforcement learning. *arXiv preprint arXiv:1903.02020*.

David Ha and Jürgen Schmidhuber. 2018. World models. *arXiv preprint arXiv:1803.10122*.

Jonathan Ho and Stefano Ermon. 2016. Generative adversarial imitation learning. In *Advances in neural information processing systems*, pages 4565–4573.

Diederik P Kingma and Max Welling. 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

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

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

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

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

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

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359.

Xin Wang, Qiuyuan Huang, Asli Celikyilmaz, Jianfeng Gao, Dinghan Shen, Yuan-Fang Wang, William Yang Wang, and Lei Zhang. 2019. Reinforced cross-modal matching and self-supervised imitation learning for vision-language navigation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6629–6638.