Reward Modeling from Human Preferences and Advantage Actor-Critic Reinforcement Learning: A Reproducibility Study

Ian Randman
Rochester Institute of Technology
Rochester, NY 14623
`ixr5487@rit.edu`

David Dunlap
Rochester Institute of Technology
Rochester, NY 14623
`dad9489@rit.edu`

Abstract

Many artificially intelligent systems in use today utilize a method called reinforcement learning, where the agent learns which behaviors are good, and which are bad, similar to how a child receives punishments and rewards from parents to reinforce certain behavior. In practical use, it is difficult for the agent to learn complex behaviors, as it is difficult to programmatically define what good and bad behavior looks like. This paper seeks to validate the method from Christiano et al. [2017], to train agents in OpenAI Gym using deep reinforcement learning and reward modeling. Complex behaviors are learned using a human to give feedback on which of two trajectories exhibits more preferable behaviors. This method requires human feedback on only a small portion of the total agent interactions with the environment. Using human feedback is generally a good method to tackle the agent alignment problem, where an agent exploits flaws in a reward function and does not align with the expectations of the person who developed the model. The use of preferences over an alternate method like a simple feedback scale for a single trajectory reduces the difficulty of the task for the human, as it is difficult for a human to assign a precise number to such an abstract goal. Our results, while evaluated on much simpler tasks than that of Christiano et al. [2017], do suggest that using human preferences over trajectories is valid for assisting an RL agent to learn complex tasks.

1 Introduction

Reinforcement learning uses the idea of assigning a reward to an agent based on the actions it takes in a given situation. The agent can then develop a policy to maximize this reward. In practice, several issues arise when developing a reward function. The most apparent problem is that it is often difficult to write the reward function. Doing so requires technical knowledge about the domain, and even then, the function may not encapsulate all aspects of the task that matter for the agent to perform "well." It is often easier to criticize a model than it is to develop the model in the first place. For example, it is easier to tell a good backflip from a bad one than it is to describe programmatically how to do a backflip. In essence, "you know it when you see it." For this, we introduce the concept of reward modeling, where the reward function is learned with the help of human feedback. Ultimately, the reward function will be modeled in part from the user's intentions. Doing so directly addresses the agent alignment problem, where an agent will exploit an action that is viewed positively by the reward function but may not be in alignment with the user's intentions, and this plays a large role in AI safety as well.

For the agent to then create a policy that maps its understanding of the reward function and the current state to a move has its own challenges. This is the core of the reinforcement learning agent, for which we introduce the technique of Advantage Actor-Critic (A2C). This is a reinforcement learning technique based around an actor-critic model, where the model outputs both a policy (the actor) and a value (the critic). The policy will give a probability distribution to describe what reward it expects to receive by taking each of its available actions, and the value output scores the potential reward that can be expected given the current state. Using each of these outputs, the model can efficiently learn a concept of the reward function and be able to take actions that maximize its reward.

With the combination of these methods, with a reinforcement learning agent implemented with A2C taking actions in its environment and receiving rewards from the reward model ANN trained using supervised learning on human feedback, we can expect to have an agent that behaves in a way that accurately reflects the human's preferences on its behavior, without having to explicitly specify a reward function.

2 Related Work

Before this method, there were many papers written about using human preferences in place of a reward function both for reinforcement learning agents and traditional agents. There are a few papers that have been written that use a very similar method to the paper we are reproducing, Christiano et al. [2017].

Our paper can be seen somewhat as an extension of Hadfield-Menell et al. [2016], where they describe a game between a human and an agent, where the human and the agent are both interacting with the environment and the goal is to maximize the human's internal reward function. Our method is very similar, but in our environment, the human can't directly interact with the environment. Instead, they provide feedback on the agent's performance which is used for learning. Mnih et al. [2016] describes the A2C/A3C method used to implement our reinforcement learning agent. In addition, our overall method is described on a higher level in Ibarz et al. [2018] and Leike et al. [2018]. Ibarz et al. [2018] describes a reward modeling system for ATARI games, where the agent is trained using reward modeling and expert demonstration. This reward modeling system is very similar to ours, but we do not use expert demonstration – all of our agents are trained from non-expert feedback. Leike et al. [2018] Describes the concept of reward modeling on a higher level and discusses some of the potential benefits introduced – namely agent alignment.

There are also some papers that use the same basic approach as our method but differ in some key areas. Akrour et al. [2012] and Schoenauer et al. [2014] describe a method of using human feedback to create a reward model, but they focus strictly on a continuous domain with smaller degrees of freedom than we explore. They also collect human feedback from entire trajectories instead of short clips, meaning it takes longer to give feedback and less human preferences can be collected. Wilson et al. [2012] is also similar, but they use a Bayesian model for the reward function instead of human feedback. They structure their loss similarly, modeling preferences, but their feedback is synthetically drawn from the Bayesian model. Finally, MacGlashan et al. [2017] and Pilarski et al. [2011] take a similar approach, but their model can only be trained when a new human feedback preference is provided. This means the system takes much longer to learn than ours, which can train alongside the human giving feedback.

Aside from our method, there are many papers that use human feedback to tune the policy of a reinforcement learning agent. Finn et al. [2016] explores reinforcement learning in a semi-supervised environment, where the human occasionally provides feedback in some controlled setting, but then the model is left to train and explore on its own. Fürnkranz et al. [2012] describes a method of using human feedback to generally guide the model, but not train it implicitly. They collect qualitative feedback from the human such as ranking the agent's available actions and train the model to take this ranking into account. Singh et al. [2019] trains the agent entirely with examples as opposed to a reward function. It gets feedback from a user when it successfully completes a task, and is structured so that feedback is only needed for a fraction of the states.

Finally, there are a few papers that are tangentially related, in that they focus on learning from human feedback, but not necessarily in a reinforcement learning setting. One of these is Sørensen et al. [2016] in which users select from videos of an agent playing Super Mario and these selections are used to train the model. This is not a reinforcement learning environment, but their method of attaining human feedback is similar to ours. Ho and Ermon [2016] describes another method of learning from humans – imitation learning. Here, the agent learns a policy from example expert behavior using a generative adversarial structure. While this method is a good way to train based on human preferences, it requires the human to demonstrate the task. Our method is applicable to all tasks that a human is able to evaluate, even those that are difficult to demonstrate.

3 Problem Description

Reinforcement learning is often the preferred method of getting an artificially intelligent agent to perform a complex task. To create a reward function for this task, we use a technique called reward modeling, described in Section 3.1. The alternative to reward modeling is to write an objective reward function, which can be quite difficult. The RL algorithm we use is Advantage Actor-Critic, described in Section 3.2.

3.1 Reward Modeling

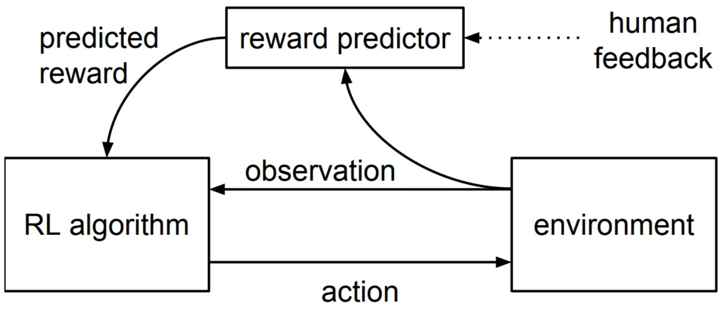

Figure 1: Illustration of the reward modeling system. The reward predictor is trained on human feedback and used to evaluate the RL algorithm.

Over the years, it has been proven that the real challenge in reinforcement learning research is not to build an agent that can effectively optimize a given reward function, but to build the reward function in the first place. Very often the agent quickly learns to optimize the function, but it becomes apparent that there is a key exploitable flaw in the function that allows the agent to game the system. This can happen in many unpredictable ways, including the agent exploiting a bug in the simulated environment itself.

Reward modeling aims to solve this core issue by having the agent learn the reward function. Instead of having a handwritten reward function that is used to evaluate the model, this function becomes an agent itself, which is trained on human feedback to help determine what reward the agent should receive at any given time-step (see Figure 1). The reward agent can be trained in many ways, as long as it is trained to attempt to model the human's true reward function. One approach is to have it evaluate two different agents' trajectories, or series of actions over a given time, and determine which agent is behaving more like the human wants it to [Christiano et al., 2017]. This agent can then be trained using a simple supervised learning approach, where the model provides the images to the human for feedback and predicts which one the human will choose. Training can be sped up even more by having the agent give the human the examples where it is most unsure which agent is performing better, thus allowing the agent to quickly perfect its flaws and uncertainties. Using this model, the reinforcement learning agent will be able to learn to complete any task that the reward modeling agent can internalize. For example, it can learn to not only make an agent that can walk, but it can also train it to hop on one leg. By selecting for certain behavior, the human can have granular control over the exact behavior of the agent.

Reward modeling also allows us to get around the credit assignment problem, where it is unclear how to attribute scores to specific actions taken in the past. This is solved by having an agent take care of this issue instead of the handwritten reward function. Reward modeling abstracts away many difficulties of reinforcement learning in this way, by just having the agent take care of this in its model of the human's intentions. Based on success with other fuzzy tasks such as labeling images of cats and dogs, we can expect to have success with developing a model that can capture the fuzzy concepts required to understand the user's true intentions.

3.2 Advantage Actor-Critic (A2C)

There are two primary methods when it comes to reinforcement learning: value-based methods and policy-based methods. Value-based methods map state-action pairs to values (in the case of Deep

Q-Learning, there is a NN that takes in a state and outputs a value and an action). Policy-methods optimize a policy that maps states to actions. Value-based methods start to show their weakness in real-world applications. A significant drawback is the value function enumerates over all possible actions. However, for tasks such as driving a car, the action space for turning the steering wheel is continuous, making a value-based approach infeasible. Policy-based methods also have a glaring fault: policy gradient waits until the end of an episode to calculate a reward. If there is high reward at the end of the episode, a conclusion may be made that all the actions taken were good. In reality, it is possible for there to exist a bad action in the sequence that still leads to a high total reward. This false conclusion may lead to slow learning, because it thinks that some bad actions are actually good.

The actor-critic model is a hybrid of value-based and policy-based methods. It is composed of the actor (policy-based) and the critic (value-based). As is common in human behavior, it is beneficial to reflect on performance and observations at shorter time intervals, rather than waiting until the end, to not miss any important events. The actor selects actions, and the critic evaluates the actions taken by the actor. Let's say the actor is playing a video game. At the start, the actor is clueless about how to play, so it will take a random action. The critic will observe this action and provide feedback. The actor will then update its policy according to this feedback, so it will be better at playing the game. The critic will also update how it provides feedback based on observations.

Another pitfall of value-based reinforcement learning is high variability for when small changes are made to the value function. To mitigate this issue, an advantage function is used in place of the value function. This function communicates the improvement that taking an action at a state gives compared to the expected value of that state.

To implement the actor and the critic, two neural networks are used. The actor follows a policy $\pi(a_t|s_t;\theta)$ to decide actions given a state. The critic maintains an estimate of the value function $V(s_t;\theta_v)$. Training these networks occurs as follows:

> The policy and the value function are updated after every t_{max} actions or when a terminal state is reached. The update performed by the algorithm can be seen as $\nabla_{\theta'} log\pi(a_t|s_t;\theta')A(s_t,a_t;\theta,\theta_v)$ where $A(s_t,a_t;\theta,\theta_v)$ is an estimate of the advantage function given by $\sum_{i=0}^{k-1}\gamma^i r_{t+i}\gamma^k V(s_{t+k};\theta_v) - V(s_t;\theta_k)$, where k can vary from state to state and is upper-bounded by t_{max}. [Mnih et al., 2016]

A2C can also be configured to use of multiple agent copies to explore the environment that will synchronously send their back findings to a shared model.

4 Our Method

We seek to validate the methods from Christiano et al. [2017], to train agents in OpenAI Gym using deep reinforcement learning and reward modeling. The A2C method, instead of AC3, will be used to train the reinforcement learning agent. This is just for ease of training. The reward function will be learned using reward modeling with human feedback, where a human selects which of two demonstrations (a.k.a. segments, which is a portion of a trajectory) is more favorable. We will compare the results from our implementation of a reward model with results from using the rewards that are provided by the OpenAI Gym environments. Both will use our implementation of the Advantage Actor-Critic RL model.

4.1 Goal

As per Christiano et al. [2017], a trajectory segment σ is a sequence of observations and actions, $\sigma = ((o_0,a_0),(o_1,a_1),...,(o_{k-1},a_{k-1})) \in (\mathcal{O}\times\mathcal{A})^k$, where $o_t \in \mathcal{O}$ is an observation from the environment, and $a_t \in \mathcal{A}$ is an action that is sent to the environment at each time step t within a trajectory. We say that σ^1 is preferred to σ^2, $\sigma^1 \succ \sigma^2$, whenever $r(o_0^1,a_0^1)+...+r(o_{k-1}^1,a_{k-1}^1) > r(o_0^2,a_0^2)+...+r(o_{k-1}^2.,a_{k-1}^2)$, such that r is the reward function, $r:\mathcal{O}\times\mathcal{A}\to\Re$.

We use these preferences to train the reward model to better predict actual rewards, which are fed into the A2C model to perform regular deep reinforcement learning. To do this, we first calculate the probability of choosing one segment over another:

$$\hat{P}[\sigma^1 \succ \sigma^2] = \frac{\exp \sum \hat{r}(o_t^1, a_t^1)}{\exp \sum \hat{r}(o_t^1, a_t^1) + \exp \sum \hat{r}(o_t^2, a_t^2)}. \tag{1}$$

Then, $\hat{r}$ is chosen to "minimize the cross-entropy loss between the predictions and the actual human labels" [Christiano et al., 2017]:

$$loss(\hat{r}) = - \sum_{(\sigma^1, \sigma^2, \mu) \in \mathcal{D}} \mu(1) log \hat{P}[\sigma^1 \succ \sigma^2] + \mu(2) log \hat{P}[\sigma^2 \succ \sigma^1] \tag{2}$$

,

where μ is a distribution over $\{1, 2\}$ indicating the preferred segment. Each pair of segments, along with their preference label, is stored as a 3-tuple in a database $\mathcal{D}$. The user can either select a preferred segment or that the segments are equally preferable, in which case μ is uniform.

4.2 Architecture Design

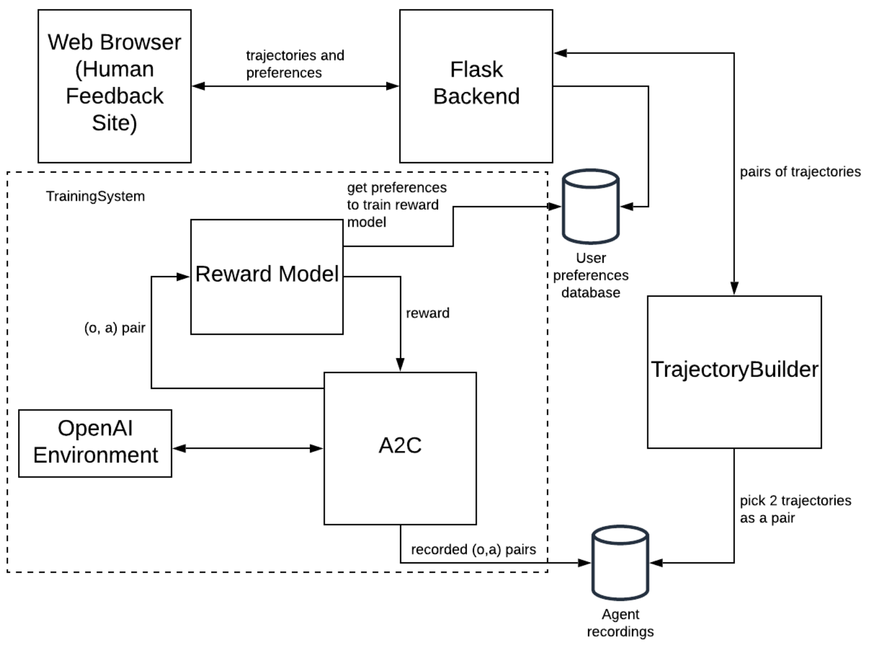
Web Browser (Human Feedback Site)
trajectories and preferences
Flask Backend
pairs of trajectories
TrainingSystem
get preferences to train reward model
User preferences database
Reward Model
(o, a) pair
reward
TrajectoryBuilder
OpenAI Environment
A2C
pick 2 trajectories as a pair
recorded (o,a) pairs
Agent recordings

Figure 2: Diagram of the design of the system as a whole. This encompasses everything from the web page for user feedback to the training of the reinforcement learning model.

In order to have a comprehensive way to collect human feedback while also training the models, we have structured our system as shown in Figure 2. We designed a web API using Flask to handle communication between the web page for selecting preferences and the rest of the system. The user preferences are collected and mapped to the two segments that were compared, which are then saved to a database to be later used to train the reward model. The TrainingSystem contains the interaction between all parts of the AI system, facilitating the training of both the reward model and the A2C agent. Pairs of (σ_t, a_t) are created, containing the data for each step. These are grouped into segments of 25-100 steps, representing 25-100 frames of video to be shown to the user. These are stored in the recording database, along with a video file of the recording. The TrajectoryBuilder handles the communication between the API getting preferences from the user and the TrainingSystem generating segments. It grabs segments from the recording database, serializes them, and packages them in JSON to be sent as an API response. This architecture allows the whole system to both run the API and train a unique A2C agent and corresponding reward model for each environment.

5 Experimental Design

We evaluated our algorithm on a few different environments provided by OpenAI Gym, primarily CartPole-v1 and LunarLanderContinuous-v2. The Cart Pole environment contains a car with a pole on a hinge. The cart can move left or right and must keep the pole from falling over. The provided reward function optimizes for keeping the pole as vertical as possible, and we optimized for the same behavior when giving feedback. The second environment we used was the Lunar Lander environment. Here, the agent is a robot landing on the moon, and has two rockets it can fire - one that fires left/right, and one that fires down. The provided reward function optimizes for the agent slowing its descent and landing between two flags on the ground, all while using as little fuel as possible. However, in order to see different behavior from the reward modeling system, we gave feedback indicating that the agent should hover above the ground, staying upright without landing for as long as possible.

We used these specific environments in order to explore different types of problems. Cart Pole uses a discrete action space, only able to select whether to move left or right. In contrast, we used the continuous version of Lunar Lander where it is able to fire its rockets at different speeds. Overall, the models' performance in these environments can serve as a demonstration for how it would perform in other types of environments.

6 Discussion

We had some technical difficulties with training our reward model. We struggled to find a good learning rate, as it being too low would cause the model to get "stuck," while too high of a learning rate results in vanishing gradient and our program crashing due to NaNs in the model weights. Despite this, we were able to gather results to compare overall behavior between using the reward model versus the baseline reward feedback from the environment.

6.1 Results

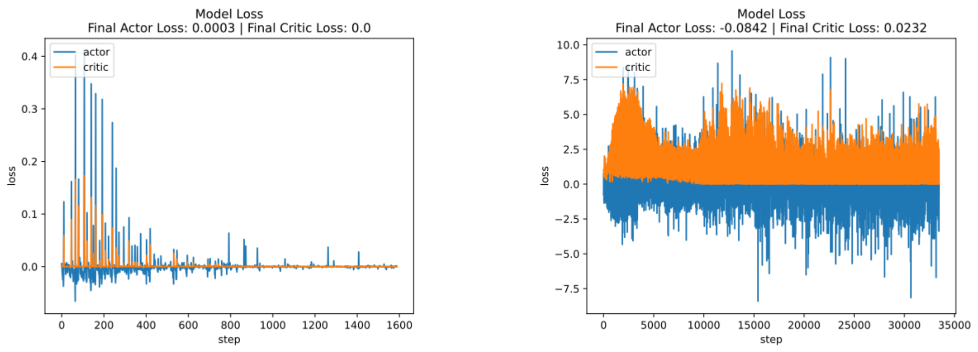

Figure 3: Cart Pole A2C training loss. Baseline reward function (Left), Reward model (Right)

In Figure 3 the training losses for the A2C agents in the Cart Pole environment are shown. For the agent that had the baseline provided reward function, Cart Pole was incredibly easy, as is demonstrated by how quickly the loss decreases on the left. However, Cart Pole proved to be a somewhat difficult challenge for a reward modeling system, since at the beginning it dies almost instantly every time. This results in the user never seeing examples of it doing any better, and has a hard time figuring out what to do. As is shown on the graph to the right, it takes roughly 4000 steps for the critic loss to stop increasing. This is most likely because the user was not able to provide feedback on the actual desired behavior.

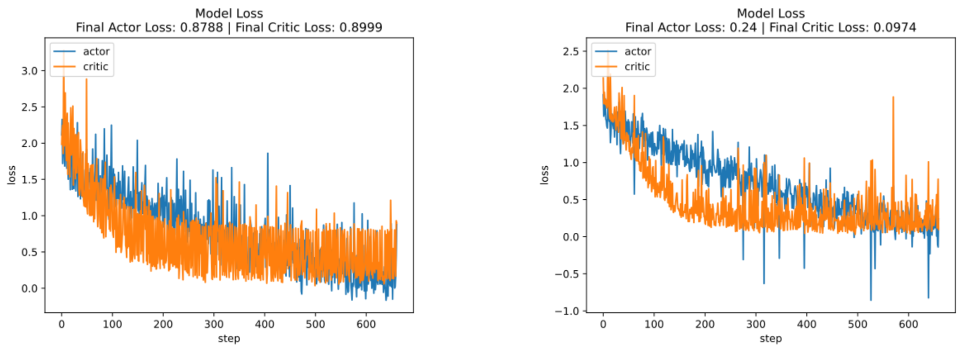

Figure 4: Lunar Lander A2C training loss. Baseline reward function (Left), Reward model (Right)

In Figure 4 the training losses for the A2C agents in the Lunar Lander environment are shown. We had some difficulty tuning the model to avoid vanishing gradient, so we were not able to run this agent for as long as the Cart Pole agent. However, we were still able to receive clear results as shown above. Both of the actor agents seem to improve linearly, while the critics seem to have a big improvement at first and then taper off, possibly indicating that they have a better sense of when they will be getting a reward. It is likely that if it were to run longer, the actor would begin to level off as well once it gets closer to 0 loss.

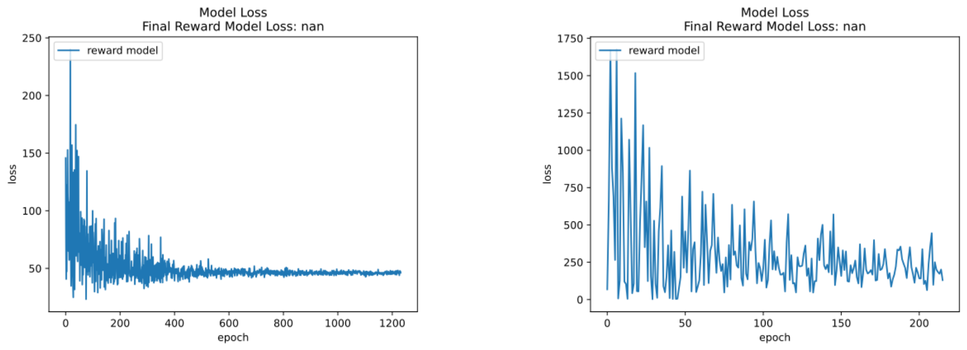

Figure 5: Reward Model training loss. Cart Pole (Left), Lunar Lander (Right)

In Figure 5 the training losses for the reward model agents are shown. The agent is clearly able to learn some sense of the human preference function, since the loss decreases significantly at the start. The shape of both graphs is relatively similar, save the shorter training length of Lunar Lander due to vanishing gradient. However, both graphs begin to level out at a relatively high loss, most likely due to the low learning rate.

6.2 Additional Insights

Our model struggles with environments that require a clear sequence of "good" actions. This is because the agent struggles to explore, and only segments from the start of the episode are presented to a human for feedback. For a clearer understanding of the problem, consider the OpenAI Gym environment Mountain Car. In this environment, the agent is a car at the bottom of a valley and needs to reach the top of the hill. In order to get up the main hill, the car must first gain momentum by going up the smaller hill behind it. If the car never tries that to begin with, the human will never be presented with demonstrations where the state of the car is up the hill. Environments like Cart Pole generally are a bit easier, because there is no natural sequence to the episode. Once it figures out how to balance, the agent can just keep the pole there, because the subsequent observations are very similar to each other.

However, for an environment like Cart Pole to do better, it needs to first have had a few correct actions to get to the equilibrium state. During our training, Cart Pole struggled to stop the pole from immediately falling, and so the episode ends. This means that the demonstrations shown to the human are very similar, and so the reward function outputs for the observation space close to the start of the episode are always changing. This is why the A2C loss alternates between decreasing and increasing for Cart Pole. In comparison, the episodes of Lunar Lander do not end until it crashes, which happens after many more steps than that of Cart Pole. This means that the human is presented with segments that look much more different each time. Consequently, the reward model for a given sequence of observation-actions does not change much, as the feedback is more sparse, resulting in a more consistent direction of the loss graph.

7 Conclusion

Our results show that, given enough accurate human preference data, a reinforcement learning agent is able to preform well using rewards assigned by a reward model that was trained on this preference data. It is also able to come to a suitable solution with relatively minimal human interaction. We were not able to recreate the results as accurately as Christiano et al. [2017], but this seems to be mostly due to difficulties with tuning some of the hyperparemeters such as the learning rate. If our implementation were more robust to hyperparemeters, we feel confident that we could produce results similar to those achieved by Christiano et al. [2017].

7.1 Future Work

Our project focused on validating the method of Christiano et al. [2017]. Instead of using a predefined reward function as part of reinforcement learning, we have the reward function be a neural network that learns from human preferences over two demonstrations of the agents. We used basic environments provided by OpenAI Gym to validate this method. These environments were limited to those with small action and observation spaces, as a proof-of-concept.

We would like to work on expanding to more complex environments. This includes tools like MuJoCo for continuous control task and ATARI games. We plan to generalize our model to be more robust to the given environment, incorporating image processing so as to not need predefined features. Using environments outside of OpenAI Gym, our model will be applied to real-world problems for which modern reinforcement learning is applied.

Different RL algorithms, such as Deep Q Learning over A2C will be compared to evaluate how they react to ongoing changes in the reward function. An alternate to A2C, Asynchronous Advantage-Actor Critic (A3C) will be focused on in order to explore a larger state-action space more quickly. In addition, different architectures and techniques of our existing networks will be experimented with. This is because one problem we encountered was not enough exploration, resulting in the model sometimes never giving the human a chance to prefer a completely opposite strategy. Good results is currently highly dependant on a good initialization of the model parameters.

Our implementation also used Keras with a backend of Tensorflow. Using Keras limits our ability to fine-tune the architectures of our models as much as raw Tensorflow would. Also, we ran into lots of issues because of the fact that Tensorflow uses a static computational graph. We would like to move our application to PyTorch for its intuitive APIs, the ability to fine tune, and the usage of a dynamic computational graph.

References

Riad Akrour, Marc Schoenauer, and Michèle Sebag. April: Active preference-learning based reinforcement learning, 2012.

Paul Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences, 2017.

Chelsea Finn, Tianhe Yu, Justin Fu, Pieter Abbeel, and Sergey Levine. Generalizing skills with semi-supervised reinforcement learning. *CoRR*, abs/1612.00429, 2016. URL `http://arxiv.org/abs/1612.00429`.

Johannes Fürnkranz, Eyke Hüllermeier, Weiwei Cheng, and Sang-Hyeun Park. Preference-based reinforcement learning: A formal framework and a policy iteration algorithm. *Machine Learning*, 89(1-2):123–156, 2012. ISSN 0885-6125. doi: 10.1007/s10994-012-5313-8. Special Issue of Selected Papers from ECML/PKDD-11.

Dylan Hadfield-Menell, Anca D. Dragan, Pieter Abbeel, and Stuart J. Russell. Cooperative inverse reinforcement learning. *CoRR*, abs/1606.03137, 2016. URL `http://arxiv.org/abs/1606.03137`.

Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *CoRR*, abs/1606.03476, 2016. URL `http://arxiv.org/abs/1606.03476`.

Borja Ibarz, Jan Leike, Tobias Pohlen, Geoffrey Irving, Shane Legg, and Dario Amodei. Reward learning from human preferences and demonstrations in atari. *CoRR*, abs/1811.06521, 2018. URL `http://arxiv.org/abs/1811.06521`.

Jan Leike, David Krueger, Tom Everitt, Miljan Martic, Vishal Maini, and Shane Legg. Scalable agent alignment via reward modeling: a research direction. *CoRR*, abs/1811.07871, 2018. URL `http://arxiv.org/abs/1811.07871`.

James MacGlashan, Mark K. Ho, Robert Tyler Loftin, Bei Peng, David L. Roberts, Matthew E. Taylor, and Michael L. Littman. Interactive learning from policy-dependent human feedback. *CoRR*, abs/1701.06049, 2017. URL `http://arxiv.org/abs/1701.06049`.

Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016. URL `http://arxiv.org/abs/1602.01783`.

P. M. Pilarski, M. R. Dawson, T. Degris, F. Fahimi, J. P. Carey, and R. S. Sutton. Online human training of a myoelectric prosthesis controller via actor-critic reinforcement learning. In *2011 IEEE International Conference on Rehabilitation Robotics*, pages 1–7, June 2011. doi: 10.1109/ICORR.2011.5975338.

Marc Schoenauer, Riad Akrour, Michele Sebag, and Jean-Christophe Souplet. Programming by feedback. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 1503–1511, Bejing, China, 22–24 Jun 2014. PMLR. URL `http://proceedings.mlr.press/v32/schoenauer14.html`.

Avi Singh, Larry Yang, Kristian Hartikainen, Chelsea Finn, and Sergey Levine. End-to-end robotic reinforcement learning without reward engineering. *CoRR*, abs/1904.07854, 2019. URL `http://arxiv.org/abs/1904.07854`.

P. D. Sørensen, J. M. Olsen, and S. Risi. Breeding a diversity of super mario behaviors through interactive evolution. In *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–7, 2016.

Aaron Wilson, Alan Fern, and Prasad Tadepalli. A bayesian approach for policy learning from trajectory preference queries. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1133–1141. Curran Associates, Inc., 2012. URL `http://papers.nips.cc/paper/4805-a-bayesian-approach-for-policy-learning-from-trajectory-preference-queries.pdf`.