Explore, Exploit or Listen: Combining Human Feedback and Policy Model to Speed up Deep Reinforcement Learning in 3D Worlds

Zhiyu Lin, Brent Harrison, Aaron Keech, and Mark O. Riedl
School of Interactive Computing
Georgia Institute of Technology
Atlanta, GA
[zhiyulin; aaronkeech; brent.harrison; riedl]@gatech.edu

Abstract

We describe a method to use discrete human feedback to enhance the performance of deep learning agents in virtual three-dimensional environments by extending deep-reinforcement learning to model the confidence and consistency of human feedback. This enables deep reinforcement learning algorithms to determine the most appropriate time to listen to the human feedback, exploit the current policy model, or explore the agent's environment. Managing the trade-off between these three strategies allows DRL agents to be robust to inconsistent or intermittent human feedback. Through experimentation using a synthetic oracle, we show that our technique improves the training speed and overall performance of deep reinforcement learning in navigating three-dimensional environments using Minecraft. We further show that our technique is robust to highly innacurate human feedback and can also operate when no human feedback is given.

Introduction

Interactive machine learning (IML) (Fails and Olsen Jr 2003) seeks to improve upon traditional machine learning algorithms by allowing humans to play a direct role in training by providing demonstrations of correct behavior or by actively critiquing the model while the agent is learning. This has proven to especially effective at speeding up learning in complex sequential decision making environments that are often solved using reinforcement learning (RL). This is because the human feedback can be used to enable the agent to explore reasonable behavior trajectories when it would otherwise have no prior knowledge to dictate behavior.

Human feedback in IML can take different forms. *Learning from Demonstration* allows humans to directly provide examples of proper behavior(Argall et al. 2009). The agent can learn the policy directly, learn to explore more effectively (Griffith et al. 2013), or learn a reward function from which to reconstruct a policy (Abbeel and Ng 2004). It is not always feasible to provide demonstrations. *Learning from Critique* allows human teachers to indicate that the agent is doing well or not doing well in order to bias the agent toward certain outcomes. Learning from Critique can also include human indication of preferences over variations in agent behavior (Christiano et al. 2017). *Learning from Advice* is similar to Learning from Critique, except the human teacher advises the agent on the actions it should take. Preliminary experiments (in preparation) show that humans prefer giving action advice over critique. This paper looks at incorporating advice from human teachers into deep reinforcement learning.

Recently, reinforcement learning approaches augmented with deep neural networks have proven to be effective at learning policies in complex sequential environments, such as Atari games (Mnih et al. 2013) or Minecraft (Johnson et al. 2016; Oh et al. 2016), with only access to pixel-level state representations. While Deep Reinforcement Learning (DRL) is effective at learning control policies in these environments, they are data inefficient in that it can require a large amount of learning episodes and exploration to learn a reasonable policy. This often makes these techniques ill-suited on complex, real-world problems. To address this limitation, we seek to extend current deep reinforcement learning techniques to enable them to learn from discrete human feedback. By allowing DRL techniques to learn from human feedback, we seek to drastically reduce the required number of training episodes to learn a reasonable behavior policy in complex virtual worlds.

Dealing with human feedback is not a trivial task due to many factors including:

- **Inconsistency.** It is unlikely that a human teacher will be able to consistently provide correct feedback. These inconsistencies can arise because the human trainer does not have a complete grasp of how to complete the task themselves, or possibly because of fatigue or simply making mistakes.
- **Intermittent Feedback.** It is also not guaranteed that humans will provide feedback on each action that the agent takes. This means that the human reward signal provided to the agent could be very sparse, further complicating the learning process.
- **Differing scales of reward and feedback.** In the simplest interpretation, human feedback can be considered part of the reward signal. However, if there is reward emanating from the environment (a common assumption), then the scale of the environmental reward and scale of

feedback values must be tuned to achieve peak learning performance.

- **Latent states.** In DRL in 3D worlds the true state of the agent and environment must be inferred from pixel-level observations which may contain a nontrivial amount of sensor noise. Unlike discrete environments, it becomes difficult to determine which states to apply human feedback values.

In this paper, we extend off-policy DRL techniques to learn from human advice while taking into account that this advice may be inconsistent and intermittent. Off-policy reinforcement learners balance exploitation of policy and pure exploration by occasionally selecting a random action. Whereas most IML has explored discrete environments—often 2D games—we further show that Learning from Advice can work in 3D worlds where environmental reward and human feedback values are on different scales. We evaluate our technique in the three-dimensional virtual world, Minecraft, using a set of simulated oracles meant to mimic human trainers of varying training proficiencies.

Preliminaries

A Markov Decision Process (Puterman 2014) (MDP) is a model used to describe potentially stochastic sequential decision making problem.

A MDP can be expressed as a tuple $\langle S, A, R, T \rangle$ which contains:

- A set of possible world states S
- A set of possible agent actions A
- A reward function $R(s, a) : S \times A \rightarrow \mathbb{R}$
- A transition function $Pr(s'|s, a) : S \times A \times S' \rightarrow p \in [0, 1]$ of each action's possible effects in each state.

The solution to a MDP is a policy $\pi : S \rightarrow a$, which is a function that dictates the best action an agent can take in any world state in order to maximize future rewards. Reinforcement Learning is a technique that learns an optimal policy for a MDP by stochastically performing actions and observing their effect on the world.

Q-Learning (Watkins and Dayan 1992) is one of the approaches in RL to help AI agents to approximate a reward function. An estimate of state-action values, $Q(s, a)$, is iteratively updated in each learning phase as follows:

$$Q(s,a) = Q(s,a) + \alpha[R(s) + \gamma \max_{a'} Q_{old}(s', a') - Q(s,a)] \quad (1)$$

where γ is a predefined discount factor and α is the learning rate.

For large or unknown environments, Deep Q-Learning (Gu et al. 2016) is an extension of original Q-Learning that utilizes a deep neural network to approximate the $Q(s, a)$ even when the number of states s is large.

Related Work

There has been much work done on incorporating demonstrations (Schaal 1997; Wilson, Fern, and Tadepalli 2012; Wirth et al. 2016; Akrour, Schoenauer, and Sebag 2011) and critique (Argall et al. 2009; Wilson, Fern, and Tadepalli 2012; Daniel et al. 2015; Wirth et al. 2016; Christiano et al. 2017) into machine learning. These approaches have proven effective at speeding up learning in complex, sequential environments. Typically these methods assume the existence of a reward function and use human feedback to aid the agent in learing a policy that maximizes that reward. Inverse reinforcement learning (Abbeel and Ng 2004; El Asri et al. 2016), on the other hand, seeks to directly engineer a reward function based on examples of optimal behavior provided by human trainers.

Video games are complex virtual worlds that often emulate many of the complexities found in the real world. Thus, many machine learning researchers have taken an interest in using machine learning to train AI agents to play video games (Mnih et al. 2013; Griffith et al. 2013). So far, there have been successes in using machine learning in both 2-D (Mnih et al. 2013; Mnih et al. 2015) and 3-D environments (Griffith et al. 2013; Oh et al. 2016; Christiano et al. 2017).

There are studies focusing on combining the reward learning methods with human input, such as (Judah et al. 2010; Griffith et al. 2013). We seek to extend this work in this paper, especially the work performed in (Griffith et al. 2013). Their method aims to rescale the human feedback and generate a universal value. We extended it by utilizing probabilistic approaches and deep reinforcement techniques to enable similar methods to be adapted to continuous state space.

A noticeable difference between our study with others is that ours are based on Deep Q-Learning. Instead of a discretized state space, our assumption is that the state space can be continuous thus a finite array of states can not be easily defined. Since Deep Q-Learning is a relatively new topic compared to the long history of Learning from Demonstration, many techniques are based on assumptions that no longer stand in 3-D environments and we are unable to apply their methods directly nor compare our method with theirs.

In the context of incorporating human feedback into a deep reinforcement learning paradigm, Christiano et al. (2017) collects preferences over action trajectories and trains a neural network to produce reward values. This is similar to the work by Wilson, Fern and Tadapalli (2012) which uses preferences over trajectories in Bayesian policy learning. Though the human feedback gathering interface of our methods and theirs share similar traits, our method has fundamental differences with theirs. Our work differs from these approaches in that we combine environmental reward and human feedback instead of relying only on preference feedback. Video games have clear rewards and the goal of our work is to augment agent ability to learn a policy in environments where it is difficult to learn an optimal policy even with the presence environmental rewards. We believe that our method and theirs can be combined to achieve better performance and reduce the feedback query frequency, but we leave it to future work.

Method

The challenges of incorporating human feedback into deep reinforcement learning are two-fold. First, we must co-

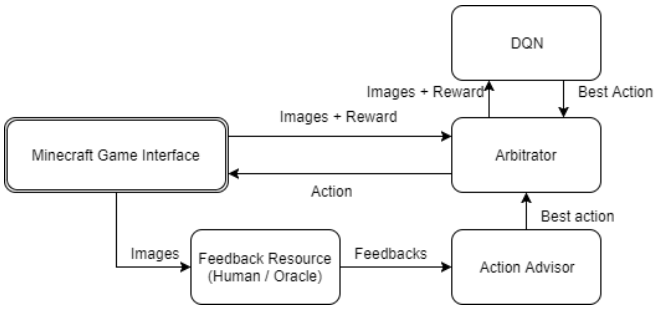


Figure 1: The agent architecture.

struct a deep neural network that learns to map pixels from an agent's sensors into Q-values—called a Deep Q Network (DQN). Much of prior work on interactive machine learning that incorporates human critique feedback (Griffith et al. 2013) was performed in discrete environments with no sensor error. In these environments, states can be uniquely matched when determining how to adjust the Q-values of states based on human feedback.

Second, if there is a reward signal from the environment then scaling the human feedback and/or environment reward appropriately is a non-trivial tuning problem. An environmental reward signal that is strong relative to the human feedback may make an agent incapable of learning from human feedback. This is a well-known problem in *modular reinforcement learning* (Simpkins 2010). A large amount of feedback may also overwhelm environmental rewards. Further complicating the situation, a human trainer may not choose to provide feedback at every state. It is thus also non-trivial how to handle "silence". However, even the naive solution of rendering lack of feedback as a zero can be problematic when the environmental reward scale allows for negative reward values.

In this section, we describe a technique for incorporating human feedback into a deep reinforcement learning paradigm. Our technique makes use of an *arbiter*, which decides when to execute an action computed by a deep Q network or to follow human action advice (Figure 1). The arbiter measures the confidence in the deep Q network as it learns—a function of network loss—and the consensus between the human action advice and the action selected by the deep Q network. The arbiter then picks between random exploration, exploiting the action picked by the deep Q network, and exploiting the human action advice (if any). Our technique is thus an off-policy reinforcement learner.

Figure 1 shows the system architecture for our technique. The Minecraft environment and deep Q network modules form the standard reinforcement learning loop—the environment produces a state and a reward, which is consumed by the learner and an action is executed in the environment. In the case of 3D virtual environments such as Minecraft, the state is an image generated from the first-person perspective of the situated agent. A second loop exists where a human oracle (or synthetic oracle for experimentation purposes) is also observing the environment state and *may* choose to provide feedback in the form of action advice. Action advice indicates which action(s) the oracle believes the agent should take. As noted above, the arbiter sits in the middle of both loops and must choose between the action computed by the DQN and the action advice of the oracle.

Deep Q Network

The Deep Q Network (DQN) is a neural network to approximate Q values for states—in this case first person perspective images. The DQN returns the action with the highest predicted Q value to the arbiter. We implemented the DQN used within Mnih et al. (2013), adapted for images from the 3D environment and for the Minecraft controls. Our agent is an off-policy learner because the DQN produces the action with the best predicted Q value based on network parameters and then the arbiter (described below) determines whether to execute the action or pick a random action instead.

Action Advisor

The Action Advisor acts as an interface between a human (or synthetic) oracle. The Action Advisor queries the oracle for each possible action, asking if it would be a good or bad action to take in the currently observed state. The reason we ask the oracle for a binary decision for each possible action is because there may be multiple acceptable actions. The Action Advisor uniformly selects a single action from the actions indicated to be good and passes it to the arbiter.

Note that it would be trivial to extend the Action Advisor to receive a non-binary preference for each action and sample an action from a weighted distribution over action preferences.

Arbiter: Aggregating Action Suggestions

In Deep Reinforcement Learning, aggregating information from multiple sources is not a trivial task. Risks exist in different point of transferring human feedbacks into the system, including the risk of human errors, inconsistency and limited exploration leading to incomplete learning space. We mitigate these risks using a probabilistic approach wherein "checks" are used decide the exploration schedule, or whether the agent should depend on its deep Q network, or listen to human oracle. The checks used in our technique consist of an: exploration check, confidence check, and consensus check.

Exploration Check If passed, this check guides the agent to do a random action; neither the DQN nor the Action Advisor is queried. The DQN without the Arbiter implements a typical off-policy exploitation strategy based on past experience. However, the exploration check makes the agent follow an epsilon-greedy strategy, forcing the agent to do exploration some percentage of the time. The likelihood of passing the exploration check decays over time. The probability of passing the exploration check is computed as:

$$p_{\text{explore}} = \begin{cases} 1 & t < r_{min} \\ e^{\ln 0.01 * \frac{t - t_{min}}{t_{max} - t_{min}}} & t_{min} \leq t < t_{max} \\ 0.01 & t_{max} \leq t \end{cases} \quad (2)$$

In experiments, we used $r_{min} = 600$ and $r_{max} = 2000$ in our experiments.

Confidence Check This check uses a measure on how confident the agent is in the suggestion of the DQN. When the confidence in the DQN is low, the Arbiter will prefer the action suggestion (if any) from the Action Advisor. When the confidence in the DQN is high, the Arbiter reduces the frequency that it requests advice from the oracle. The probability of passing the confidence check is computed as:

$$p_{\text{conf}} = \frac{-1}{\ln \sqrt{\frac{l}{l_{\max}}} - 1} \tag{3}$$

where l is the loss of the DQN and $l_{\max}$ is the highest loss observed so far.

Consensus Check This check uses a measure of how the best action from the DQN aligns with action advice from the oracle. The consensus check is used to counteract inconsistency in the oracle—human oracles are known to be noisy and their ability to consistently provide correct feedback can vary. When the oracle is inconsistent, and thus there is no consensus with the DQN from time interval to time interval, the arbiter relies more on the DQN. The probability of passing the consensus check is a function of the consensus probability at previous time steps:

$$p_{\text{cons},t} = \begin{cases} \max(1, p_{\text{cons},t-1}) * f_1 * d & a_{\text{DQN}} = a_{\text{AA}} \\ p_{\text{cons},t-1} * f_2 * d & a_{\text{DQN}} \neq a_{\text{AA}} \end{cases} \tag{4}$$

where a_{DQN} is the action suggestion from the DQN and a_{AA} is the action suggestion from the Action Advisor, and

$$d = \begin{cases} 1.001 & p_{\text{cons},t-1} < 0.5 \\ 0.999 & p_{\text{cons},t-1} > 0.5 \end{cases} \tag{5}$$

and $f_1 = 1.004$ and $f_2 = 0.998$. The values of f_1 and f_2 were found to empirically work well, though any value greater than 1.0 for f_1 and any value less than 1.0 for f_2 should work.

Algorithm 1: The arbiter algorithm used during agent training.

```
Initialize all learners and training parameters;
while training goal not achieved;
do
    Calculate p_explore, p_conf and p_cons;
    with probability p_explore do
        return a random action from possible
          actions;
    end
    else with probability p_conf and with probability
      p_cons do
        Request action suggestion from Action
          Advisor;
        return action from Action Advisor;
    end
    else
        Get action suggestion from DRL;
        return action from DRL;
    end
end
```

Experimental Setup

Minecraft provides players with an open world that they can explore and forge by utilizing a set group of tools to manipulate "blocks", which is the main building unit of the game. The agent is given the task of finding and picking up an object in different maze-like environments. Navigating the 3-dimensional landscape of Minecraft is an essential part of all tasks the player must perform in the game. The task of navigation is also easy to control in terms of difficulty for experimentation purposes and the availability of the game makes it easy to share results.

While the environment appears simpler than other 3D games, such as Doom, because of the use of blocks, the visual simplicity results in instances of "aliasing" where spaces can be nearly indistinguishable. Human feedback is especially important in situations of aliasing because the human teacher often has an intuitive understanding that different areas that appear similar may not be treated identically. With regard to other 3D environments such as Doom, Minecraft also makes much more use of non-enclosed spaces.

We have developed a Minecraft clone, implemented in Unity3D but sharing aesthetic and functional similarity with the original game. This was done to make agent integration and experimentation easier. To further control for the complexity of the environment, the agent is only allowed to face four 90-degree separated directions (North, South, East, West); However, we perturb the viewing angle randomly after each action by rotating the viewing angle by up to 2 degrees to the right or the left.

Since training in graphical 3D worlds creates a large computation and time overhead, we built a state cache system so that we don't have to run the graphical client for every trial. For each state—each block the agent can stand on for and each of the four cardinal directions the agent can face—we store a number of screenshots corresponding to different perturbations. The Minecraft clone tracks the true state of the agent and randomly selects an image to pass to the reinforcement learning agent.

We designed multiple different tasks at differing levels of difficulty (Figure 2. By identifying a task as a complex one, we looked into the opportunity cost of a non-optimal action. A task that is more complex than another one in this sense has more and deeper dead ends and less freedom of error correction.

Agents are given rewards on actions taken: They get -1 for each step taken and 100 for picking up the object and thus completing the task.

In our experiments, we used simulated oracle in the place of a human teacher. This is a standard technique (cf., (Griffith et al. 2013)) that grants us abilities to systematically manipulate feedback accuracy and test different parameters of environment and hypothetical teacher. The oracle was cre-

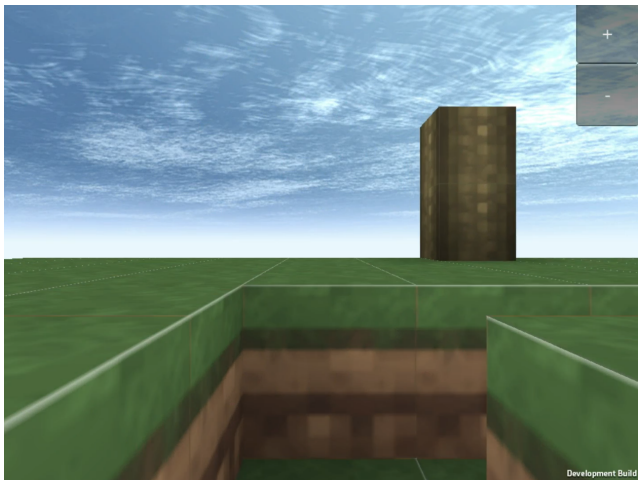


Figure 2: A sample screenshot of the agent playing.

ated by labeling each actual agent state (position, orientation) with the best action. Since human teachers can be inaccurate and have different rates of response to requests for feedback, the simulated oracle allows us to parameterize the accuracy and frequency of oracle response by randomly choosing one of possible feedback outcomes.

Experiments

Due to the nature of Deep Q-Learning, the learned policy can be very noisy (Mnih et al. 2013). However, we decided to keep that trait, using a moving average of performance since it applies to an infinite and/or loop-containing non-terminal policy, which if Q-value sums are used can lead to confusion in evaluations. After each training session is over—by finishing the task or timeout—we evaluate the agent by asking it to do the task by itself (e.g., run the policy) then record its final reward as performance.

In our experiments we compare our method with a baseline, an ablated version of our technique with the Action Advisor removed. This baseline agent is equivalent to the deep reinforcement learning technique introduced by Mnih et al. (Mnih et al. 2013) with adaptations of the input and output layers of the neural network to Minecraft. The baseline additionally uses a different epsilon decay schedule, which we empirically tuned to strengthen the baseline:

$$\epsilon = \begin{cases} 1 & t < r_{min} \\ 1 - 0.9999 * \frac{t - t_{min}}{t_{max} - t_{min}} & t_{min} \leq t < t_{max} \\ 0.0001 & t_{max} \leq t \end{cases} \quad (6)$$

where t is the training episode and t_{min} = 600 and t_{max} = 2000.

Performance with Consistent and Accurate Oracle

In the ideal case of consistent and accurate oracle feedback, Figure 4 shows that our technique converges on the optimal policy faster than the baseline. We evaluated our method on the two maps from Figure 3. Our results show that our technique never under-performs the baseline. The agent doesn't benefit from oracle feedback in the easy map because it is relatively trivial to find the optimal policy on the easy map. Two maps are sufficient to show the trend: as the map becomes more maze-like and the number of obstacles increases, the importance successfully incorporating oracle feedback into policy learning is evident.

In the case of consistent and accurate oracle feedback, the consensus check slightly reduces learning performance. Oracle feedback is considered by the Arbiter via the confidence check—if confidence in the DQN is low, seek advice from the oracle—and through the consensus check. Disabling the confidence check or consensus check affects how the agent uses the oracle feedback. The confidence check makes the most effective use of oracle feedback. This makes sense because the confidence check is guiding the agent toward the most reliable action suggestion. The consensus check, on the other hand, warns the agent away from malicious oracle action advice.

Figure 5 shows the reward as training increases for the baseline, our technique with confidence check, and our technique with both confidence and consensus checks. The figure shows the average reward, the 90th percentile (90% of all trials perform no better than this), and the 10th percentile (10% of all trials perform no better than this). The charts show that there is a very wide variance in performance for the baseline. Sometimes the learner gets lucky and hits upon a good policy, but often it times out before finding an acceptable policy. Our technique with just the confidence check or with confidence and consensus checks significantly reduces the variance in learning performance, making it much more reliably able to find the optimal policy.

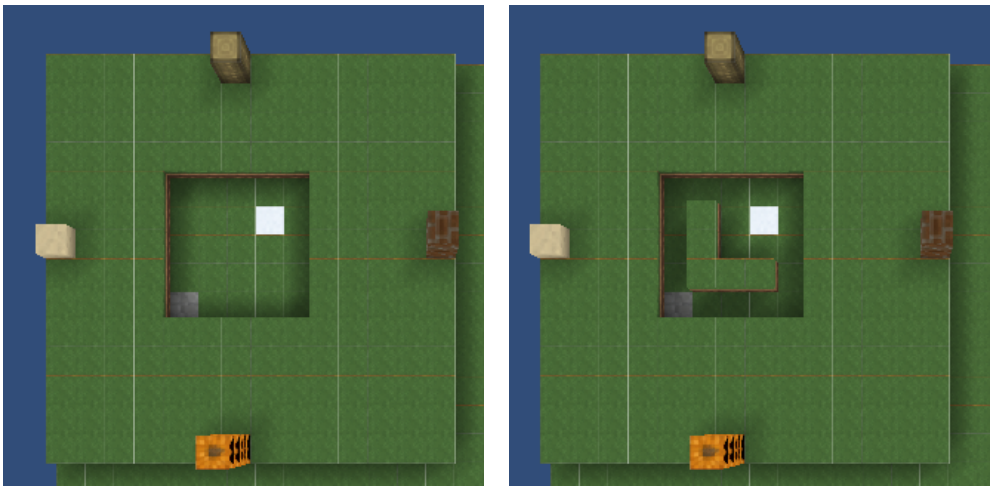
The decreased average for the consensus-only agent (Figure 4(a)) is due to a slightly increased variance—some small percentage of the time the agent will time out during trials resulting in a very low reward value being averaged in.

We also inspect the effect of our technique on requests for feedback from the oracle. Figure 7 shows the probability that the agent requests feedback from the oracle over time as training progresses. The function for random exploration (epsilon decay) is shown for reference. Note that the bulk of feedback requests come after an initial period of intense exploration and then decreases as the confidence in the DQN increases.

Performance with Differing Oracle Accuracy

We evaluated our method in non-perfect case, in which we degrade the oracle's accuracy. Figure 6 shows that the agent is robust against oracle inaccuracy. Even at 50% oracle accuracy (i.e., random feedback) our technique achieves a performance that is equivalent to the baseline. Christiano et al. (2017) assume a 10 percent of human error, well within the scope of our technique's abilities.

The extent to whether the consensus check helps is unclear. The consensus check seems to nullify the benefit of feedback as the oracle accuracy decreases. However, the average for the confidence+consistency check version is due to training timeouts and only a few timeouts can cause the average to plummet. Our synthetic non-perfect oracle naively



(a) Easy map (b) Hard map

Figure 3: Top-down view of "easy" and "hard" maps used in our experiments. The blue block is the start and the quite block is the location of the object to be picked up. Pillars are used to substitute for landscape details that partially disambiguate first-person views.

injects random advice into the agent that can be implausible, such as repeatedly advising the agent walk straight into a wall. Thus we believe that the synthetic oracle can be quite antagonistic at times, deflating the agent performance curves.

This experiment also gives an indication of how the system will respond to "silence", since lack of feedback can be interpreted as a "wrong feedback".

Conclusions

Interactive Machine Learning (IML) postulates that for environments and tasks that present substantial challenges to reinforcement learning algorithms, human teachers can help agents learn. Learning from Advice—the agent asks a human what they would do in a particular situation—has the potential to significantly improve machine learning outcomes while keeping human feedback overhead manageable. We have demonstrated that our technique for incorporating human advice in 3D graphical environments performs no worse than non-interactive learners and benefits from advice as the task becomes harder.

Two important properties of our technique are that (a) the human teacher is never required to give feedback to the agent, and (b) the human teacher is not assumed to be perfect. The former is essential for IML because providing advice has a cognitive burden on the human teacher and it may not be practical for the human teacher to weigh in on every move the agent tries while learning to perform a task. The latter is essential because humans make mistakes or can become confused themselves. We show that our technique is robust against human teacher error; as long as the human is not intentionally adversarial (i.e., greater than 50% accuracy), our agent learning technique can glean some advantage out of human feedback.

From the perspective of designing reinforcement learning agents that incorporate human feedback, our technique presents a general strategy for dealing with the fact that environmental rewards can be arbitrary and that environmental rewards and human feedback can be on drastically different scales. The arbiter is only required to decide to exploit the action suggestion of the deep Q network, exploit the action advice from the oracle, or explore the environment with a random action. It learns a schedule of exploitation, exploration, and listening to advice based on DQN learning rate and the consistency of the human oracle.

Acknowledgments

This material is based upon work supported by the Office of Naval Research (ONR) under Grant #N00014-14-1-0003.

References

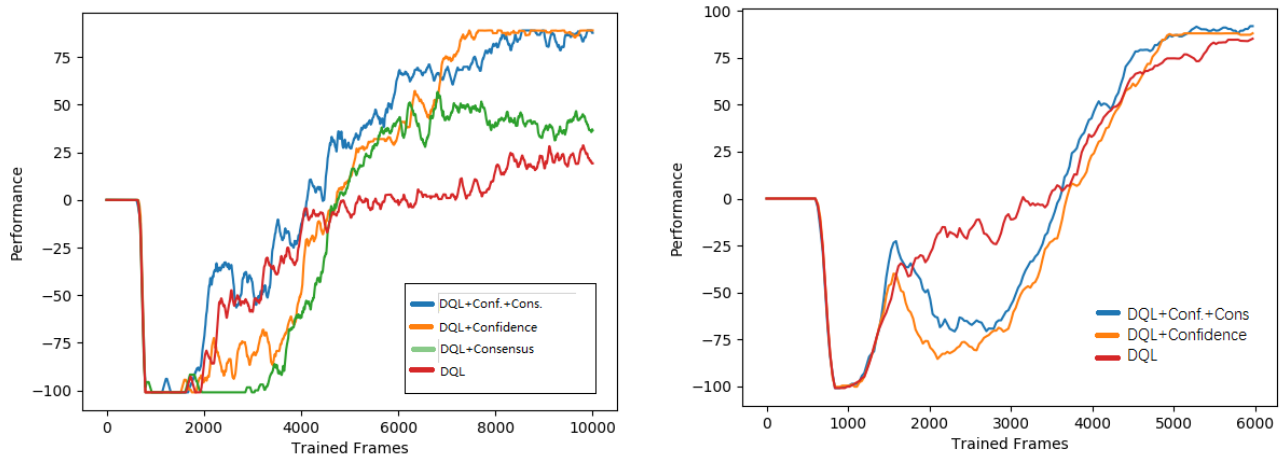
[2004] Abbeel, P., and Ng, A. Y. 2004. Apprenticeship Learning via Inverse Reinforcement Learning. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, 1–. New York, NY, USA: ACM.

[2011] Akrour, R.; Schoenauer, M.; and Sebag, M. 2011. Preference-based policy learning. *Machine learning and knowledge discovery in databases* 12–27.

[2009] Argall, B. D.; Chernova, S.; Veloso, M.; and Browning, B. 2009. A survey of robot learning from demonstration. *Robotics and autonomous systems* 57(5):469–483.

[2017] Christiano, P.; Leike, J.; Brown, T. B.; Martic, M.; Legg, S.; and Amodei, D. 2017. Deep reinforcement learning from human preferences. *arXiv preprint arXiv:1706.03741*.

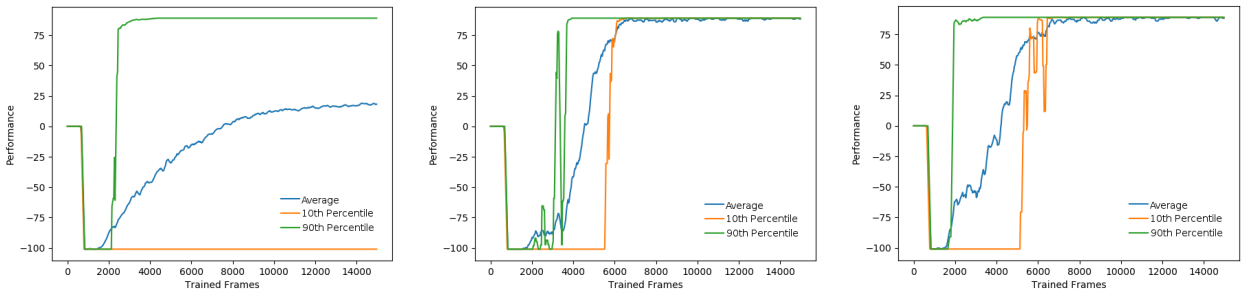
[2015] Daniel, C.; Kroemer, O.; Viering, M.; Metz, J.; and



(a) Performance on the hard map

(b) Performance on the easy map

Figure 4: Performance of our method comparing to the baseline (DQN only). Each entry represents the average performance from 20 training sessions.



(a) Baseline

(b) Condifence check only

(c) Confidence and Consistency Check

Figure 5: Performance comparison on the ”hard” map in detail.

Peters, J. 2015. Active reward learning with a novel acquisition function. *Autonomous Robots* 39(3):389–405.

[2016] El Asri, L.; Piot, B.; Geist, M.; Laroche, R.; and Pietquin, O. 2016. Score-based inverse reinforcement learning. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, 457–465. International Foundation for Autonomous Agents and Multiagent Systems.

[2003] Fails, J. A., and Olsen Jr, D. R. 2003. Interactive machine learning. In *Proceedings of the 8th international conference on Intelligent user interfaces*, 39–45. ACM.

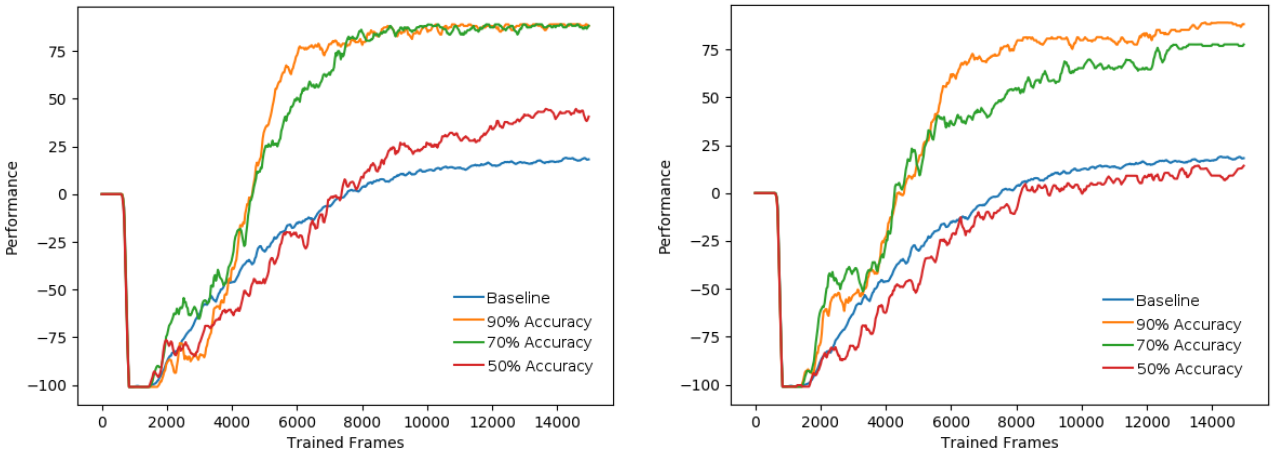
[2013] Griffith, S.; Subramanian, K.; Scholz, J.; Isbell, C.; and Thomaz, A. L. 2013. Policy Shaping: Integrating Human Feedback with Reinforcement Learning. In Burges, C. J. C.; Bottou, L.; Welling, M.; Ghahramani, Z.; and Weinberger, K. Q., eds., *Advances in Neural Information Processing Systems 26*. Curran Associates, Inc. 2625–2633.

[2016] Gu, S.; Lillicrap, T.; Sutskever, I.; and Levine, S. 2016. Continuous Deep Q-Learning with Model-based Acceleration. In Balcan, M. F., and Weinberger, K. Q., eds., *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, 2829–2838. New York, New York, USA: PMLR.

[2016] Johnson, M.; Hofmann, K.; Hutton, T.; and Bignell, D. 2016. The Malmo Platform for Artificial Intelligence Experimentation. In *IJCAI*, 4246–4247.

[2010] Judah, K.; Roy, S.; Fern, A.; and Dietterich, T. G. 2010. Reinforcement Learning Via Practice and Critique Advice. In *AAAI*.

[2013] Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.



(a) Confidence Check Only

(b) Confidence and Consensus Check

Figure 6: Performance of our method with varied feedback accuracy on different methods on ”hard” map.

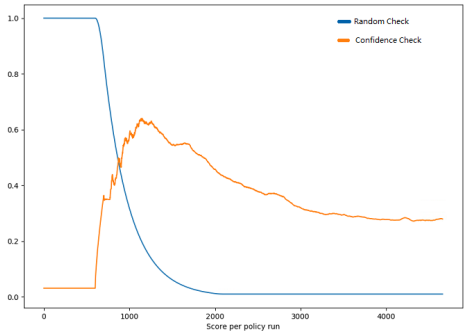


Figure 7: Probability of Random and Confidence Check passing in a typical run on ”Hard” map. Trace ends at convergence.

[2015] Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; and others. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529–533.

[2016] Oh, J.; Chockalingam, V.; Singh, S.; and Lee, H. 2016. Control of memory, active perception, and action in minecraft. *arXiv preprint arXiv:1605.09128*.

[2014] Puterman, M. L. 2014. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

[1997] Schaal, S. 1997. Learning from demonstration. In *Advances in neural information processing systems*, 1040–1046.

[2010] Simpkins, C. 2010. Integrating Reinforcement Learning into a Programming Language. In *AAAI*.

[1992] Watkins, C. J. C. H., and Dayan, P. 1992. Q-learning. *Machine learning* 8(3-4):279–292.

[2012] Wilson, A.; Fern, A.; and Tadepalli, P. 2012. A bayesian approach for policy learning from trajectory preference queries. In *Advances in neural information processing systems*, 1133–1141.

[2016] Wirth, C.; Furnkranz, J.; Neumann, G.; and others. 2016. Model-free preference-based reinforcement learning. In *30th AAAI Conference on Artificial Intelligence, AAAI 2016*, 2222–2228.