Learning good policies from suboptimal demonstrations

Yuxiang Li
Google, London, UK

Ian A. Kash
University of Illinois, Chicago, IL

Katja Hofmann
Microsoft Research, Cambridge, UK

Abstract

Imitating an expert policy is one way to boost reinforcement learning algorithms which, in most cases, rely on random exploration and a huge amount of data. While some major drawbacks of imitation learning, such as compound errors, have been addressed, most results make an implicit hypothesis of having a good (or desired) expert which is usually hard to get in practice. To overcome this obstacle, this paper focuses on learning good policies from suboptimal demonstration data. We systematically investigate the performance of a recent approach under varying assumptions on demonstration quality and show that it performs poorly with suboptimal demonstration data. We then demonstrate the potential to overcome this issue through a performance comparison between learner and demonstrator.

1. Introduction

Reinforcement Learning (RL) is a principled approach to learning from experience. It has been shown to be extremely powerful, with impressive empirical results in applications ranging from playing video games (Mnih et al., 2015) to natural language processing (Das et al., 2017). However, RL is often data inefficient, as algorithms need to try suboptimal actions many times to form accurate estimates of their consequences, which can be extremely costly if trying actions requires interaction with a complex environment. Beyond the cost, it may be unsafe to allow a poorly performing agent to interact with the actual environment.

A natural alternative is to make use of prior experience from a human expert or existing system to demonstrate acceptable policies to the learning agent (Ross and Bagnell, 2010; Monfort et al., 2017). The problem of learning a policy from such demonstration data is called learning from demonstration or imitation learning (IL). Since the ultimate goal is to use this demonstration data to get an RL agent to an acceptable level of performance, this has led to significant recent interest in approaches which combine RL and IL. However, doing so has two key conceptual challenges. First, a major reason to apply RL to a domain is that existing approaches are suboptimal, so we would like to be able to learn a better policy than is demonstrated, not just match the demonstrator’s performance. Second, the best policy available may not necessarily be the best policy for an RL+IL algorithm to learn to imitate, and the desirable properties of demonstration data are poorly understood.

In this paper we seek to understand how well current approaches tackle these challenges, and to the extent they fail what fruitful directions to improve them may be. We perform an
extensive empirical study of DQfD (Hester et al., 2018), a state-of-the-art RL+IL algorithm which has demonstrated strong performance on Atari games. To obtain demonstration data of varying qualities, we train DoubleDQN (van Hasselt et al., 2016) agents and take checkpoints at intervals in the learning process, which allows us both a high level of control over and many gradations of agent quality. We find that without careful tuning DQfD either does not benefit from or gets stuck at the performance of the demonstrator. This is illustrated in Figure 1, which shows training curves for different qualities of demonstrators and different choices of the hyperparameter $\lambda$ which represents the weight DQfD puts on its IL loss ($\lambda = 0$ is pure RL). The results are particularly stark on Breakout: low $\lambda$ values lead to optimal performance with a poor demonstrator while high ones are required with a good one. Our results provide insight into the role of exploration in demonstration data and show that the best demonstrator is not always the strongest performer. Finally, we investigate the potential of adaptively controlling the tradeoff between RL and IL. While our results are preliminary, they suggest this is a promising direction for further research.

With one exception, our experiments address the pretraining phase of an RL+IL algorithm, when it is run solely on demonstration data and unable to interact with the environment. We believe this is the crucial phase for the performance of RL+IL algorithms, as if they cannot achieve acceptable performance before interacting with the environment they may simply be infeasible. Of course it is also interesting to study the online training phase where the demonstration data is still available to improve the learning process while the agent interacts with the environment. In additional preliminary experiments (not presented) we found broadly similar results for online training to what we report for the pretraining phase. In the appendix we present one set of experiments for online training, where we examine a way of implicitly controlling DQfD’s hyperparameter which is only applicable during online training and examine the effects of its “prioritized replay” mechanism.

2. Related Work

IL comprises a range of learning approaches that make use of sample data that show the learner how to perform a given task (Argall et al., 2009; Schaal, 1999). A straightforward approach to IL is behavioral cloning, which formulates the task as a supervised learning problem where the learner maps state observations to actions, although this approach is vulnerable to compound errors (Ross and Bagnell, 2010). One solution is to mix in additional experience data (Ross et al., 2011).
Both our learners and our demonstrators use deep RL algorithms. These derive from deep Q-networks (DQN) (Mnih et al., 2015), which approximates $Q^*$ with a deep neural network, uses a replay memory, and stabilizes learning by using a target network which is only updated to match the current model occasionally. DQN has an optimistic bias, so Double DQN (van Hasselt et al., 2016) changes the update rule to mitigate this. Schaul et al. (2016) introduced a way of prioritizing data from the replay memory rather than using uniform sampling. Our work examines DQfD (Hester et al., 2018), which builds on these deep RL approaches and proposes a joint RL+IL objective. They show an increase in performance for games like Montezuma’s Revenge which were unsolvable by DQN. Using the same approach, Kurin et al. (2017) show a positive correlation between the demonstrator’s score and the learner’s score. Similar approaches have recently been used to fuse IL with other deep RL algorithms (Lakshminarayanan et al., 2016; Večerík et al., 2017).

While the work described so far assumes a (near) optimal demonstrator, other work has addressed suboptimal demonstrators. Chang et al. (2015) use a reduction to contextual bandits. Shiarlis et al. (2016) use inverse RL to learn from failed demonstrations. Brys et al. (2015) use demonstration data for reward shaping rather than as training data. Gao et al. (2018) use an RL algorithm with a loss which implicitly assumes that actions not taken in the demonstration data may be bad.

We explore the adapting the weight on the demonstration on a global basis. In parallel, Nair et al. (2017) proposed using the critic in an actor-critic algorithm to adapt on a per-state basis. Also in parallel, Schmitt et al. (2018) used trained agents as demonstrators, in their case as a way of speeding training in multi-task learning settings. They also examine tuning the demonstration weight using linear schedules and population-based approaches.

3. Preliminaries

Consider an infinite horizon discounted Markov Decision Process (MDP) $\mathcal{M} = \langle S, A, r, P, \gamma, s_0 \rangle$, where $S$ is a set of states, $A$ is a (finite) set of actions, $r(s, a)$ is the (expected) reward for taking action $a$ in state $s$, $P(s' \mid s, a)$ is the probability the next state is $s'$ given that action $a$ was taken in state $a$, $\gamma$ is the discount rate, and $s_0$ is the distribution over initial states. A policy $\pi(a \mid s)$ for $\mathcal{M}$ gives the probability of taking each action $a$ in each state $s$. Given a policy $\pi$, the expected discounted value of that policy starting from state $s$ and action $a$ can be defined as the fixed point of

$$Q^{(\pi)}(s, a) = r(s, a) + \gamma \sum_{s', a'} P(s' \mid s, a) \pi(a' \mid s') Q^{(\pi)}(s', a')$$  \hspace{1cm} (1)

The goal of value-iteration RL algorithms is to learn $Q^*(s, a) = \max_\pi Q^{(\pi)}(s, a)$, from which an optimal policy can be recovered as greedy with respect to $Q^*$.

We study the DQfD algorithm (Hester et al., 2018). During training this algorithm uses four losses: the 1-step loss from DoubleDQN (van Hasselt et al., 2016), a large margin loss, an L2 regularization loss, and an $n$-step DoubleDQN loss. To simplify the analysis, we omit the fourth loss. Unless otherwise specified we keep using their parameter values.

To give more detail on the two primary losses, the network has parameters $\theta$, takes as input $s$, and outputs $Q(s, a; \theta)$ for each action $a$. There is a replay memory $m$, which stores the demonstration data of the form $\langle s, a, r, s' \rangle$. That is, the current state $s$, the action...
taken by the demonstrator \( a \), the resulting reward \( r \), and the new state \( s' \). Minibatches of data are then sampled from this replay memory during pretraining. Letting \( \theta^- \) denote the current target network, the two primary loss functions are

\[
J_{TD}(\theta) = (r(s, a) + \gamma Q(s', a'; \theta^-) - Q(s, a; \theta))^2,
\]

(2)

where \( a' \) is the action for state \( s' \) selected as \( a' = \arg\max_{a \in A} Q(s', a; \theta) \), and

\[
J_{LM}(\theta) = \max_{a' \in A} [Q(s, a'; \theta) + \mathbb{1}_{a \neq a'}] - Q(s, a; \theta).
\]

(3)

We refer to these as the TD loss and LM loss respectively. A hyperparameter \( \lambda \) determines the weight applied to the LM loss.

To allow fine-grained control of the demonstrator, we use agents trained via DoubleDQN. For most games we trained for 15M steps, saving the network parameters every 0.5M, giving us 30 demonstrators of varying quality. For Breakout, which converges more quickly, we trained three times for 7.5M steps, giving us three distinct demonstrators of each of 16 qualities for 48 total demonstrators. Given this setup, we are then able to control, for example, the amount of data generated or the exploration rate in the training data. Note that this means we are only able to experiment on games where DoubleDQN can learn a reasonable policy within 15M steps. This notable excludes some of the more challenging games such as Montezuma, Pitfall and Private Eye. We evaluate agents by multiplying their score per episode by the number of lives. For additional experiments beyond those presented here, see the work by Li (2018).

4. Demonstrator quality

Our initial hypothesis was that the best choice of demonstrator would always be the demonstrator with the highest score. Surprisingly, this is quite far from true in some cases. Figure 2 shows the performance (averaged over 20 episodes) of DQfD on four Atari games with only the TD loss (i.e. \( \lambda = 0 \)). The \( x \) axis gives the demonstrator’s score while the \( y \) axis gives the learner’s score after 4M training steps when supplied with 500K demonstration steps gathered from the demonstrator with an exploration probability of 0.1. For Atlantis and Breakout, the best learner performance is achieved with a very bad demonstrator and the overall score of the learner is much worse than the score of the demonstrator. For Krull and MsPacman, the score of the learner gets closer to the performance of the demonstrator, but the best learner is achieved with a medium quality demonstrator. None of the games shown in Figure 2 exhibit a strong positive relationship between the 2 axes.

Figure 3 shows the results of a further set of experiments on Breakout where the exploration rate used to gather the data is varied. Figure 3a shows the training curve of the learner with a bad demonstrator (with a score of 15) with different demonstration exploration rates. The best learner is associated with a demonstration exploration rate of 0.2, so while substantial randomness is helpful, so is the structure provided by even a poor demonstrator. Figure 3b and 3c plot the same data with different \( x \) axes. Figure 3b plots the learner’s score as a function of the performance of the demonstrator if it was run with no exploration, while Figure 3c plots the score the demonstrator achieved with exploration. They show that a high exploration rate is optimal, even though this results in a poor score
for the demonstrator. Furthermore, the effects of the exploration rate are much larger than those of demonstrator quality. This suggests that one possible explanation for the superior performance of lower quality demonstrators in Figure 2 is that their decisions are somewhat more random, thus introducing implicit exploration (an idea also explored by Dauparas et al. (2018)). Another possibility, particularly in a game like Breakout which grows harder to score in over time, is that the stronger demonstrators are providing more information about late-game play, which is useless if the agent cannot survive that long.

The preceding results addressed the performance of DQfD with $\lambda = 0$. Figure 4 shows what happens when we instead take $\lambda = 1$. In contrast, there is now a clear positive and seemingly linear relationship between demonstrator and learner performance. However, as we saw in Figure 1 this comes at the cost of the learner getting stuck performing at the level of the demonstrator. In the appendix, we provide a theoretical analysis of a simplified version of DQfD to better understand this phenomenon.
5. Adaptive large margin

In this section we investigate whether it is possible to automatically achieve the best of both worlds and have the performance of $\lambda = 0$ for low quality experts and $\lambda = 1$ for high quality experts. We explore a simple, but natural idea: adjust the large margin weight based on the relative performance of the learner and the demonstrator. Doing so in practice has a problem: how can we accurately evaluate the quality of the learner given only access to data generated by the demonstrator? This is an active area of research (Dudík et al., 2014; Thomas and Brunskill, 2016), so in this work we avoid the issue by assuming a method is available. In our experiments, we evaluate the agent using our access to the environment. We believe this is reasonable given that we are more interested in the potential of this approach than proposing a particular algorithm.

Given that, we also adopt a simple heuristic for adapting $\lambda$: fix a base $\lambda_0$ and multiply it by the ratio of the performance of the learner to the performance of the demonstrator, clipping to $[0,1]$. Thus, when the performance of the learner exceeds the the imitator we are effectively in the $\lambda = 0$ case. We measure the performance of this adaptive heuristic using multiple demonstrators on Breakout and compare it with TD only and TD + LM. Results in Figure 5a confirm it has potential: in the region of sub-optimal demonstrations it achieves superior performance to either approach while in the region of high performing demonstrators it is harder to come to a firm conclusion due to high variance, but on average it seems to perform as well as TD + LM and much better than TD only.

This approach has two additional advantages. First, Figure 5b shows that we get the best of both worlds during training time as well: learning is initially fast as with TD+LM, while it avoids getting stuck and ultimately improves similarly to TD only. Second, DQfD can be sensitive to the choice of $\lambda$. In contrast, Figure 5c shows that the performance of our heuristic is relatively insensitive to the choice of its hyper-parameter $\lambda_0$, at least in Breakout. This may vary in other games, and studying it would be an interesting experiment for future work. It would be interesting to compare $\lambda_0$ tuned for a specific game versus a tuning across a broad class of games as well as using as $\lambda_0$ values of $\lambda$ that have been tuned for specific games or the generic value chosen by Hester et al. (2018).

In the appendix, we explore another heuristic for controlling $\lambda$ which is applicable during online training and provides insight into the prioritized replay mechanism used by DQfD.
6. Conclusion

We have performed an extensive empirical analysis of pretraining using DQfD, a state-of-the-art RL+IL algorithm. We have seen that the best agents are not necessarily the best “teachers”, increased exploration helps pure RL learners, DQfD tends to get “stuck” at the performance its demonstrator, and adapting the weight placed on the IL loss can help get the “best of both worlds” (and even then a suboptimal demonstrator may be best). Rather than propose a particular method to adapt the weight on the IL loss, we have explored its feasibility in a way that is on one hand infeasible in practice (because we take advantage of access to the environment which is not normally available during pretraining) but on the other extremely simple. This suggests that developing approaches which estimate only based on data actually available during pretraining but use that estimate in a more sophisticated way is likely a fruitful direction for future research.

References


Appendix A. Theoretical analysis of $\lambda$

The motivation for the use of the large margin loss is to encourage the agent to follow the actions of an expert. However, if the expert is suboptimal, this is not necessarily desirable. In this section, we provide a theoretical analysis of the consequences of using the large margin loss with arbitrary experts, and how it depends on the choice of $\lambda$.

Recall the large margin loss

$$J_{LM}(\theta) = \max_{a' \in A} [Q(s, a'; \theta) + \mathbb{1}_{a \neq a'}] - Q(s, a; \theta).$$

This is somewhat complex to analyze, as the exact penalty depends on the current Q values. Instead, we analyze this type of loss as a form of reward shaping.

In particular, the goal of our learning algorithm is to learn $Q^*$ for an MDP $M$. Consider an MDP $\overline{M}$ which is identical to $M$ except that it has a different reward function $\tau(s, a') = r(s, a') - \lambda \mathbb{1}_{a \neq a'}$. Given that $M$ and $\overline{M}$ are otherwise identical, they have the same set of stationary policies $\Pi$. Consider some policy $\pi \in \Pi$. In $M$ and $\overline{M}$, $\pi$ has Q values $Q^{(\pi)}$ and $\overline{Q}^{(\pi)}$.

**Theorem 1**

$$Q^{(\pi)} - \frac{\lambda}{1 - \gamma} \leq \overline{Q}^{(\pi)} \leq Q^{(\pi)}$$

**Proof** As Equation (1) is a contraction mapping, these values can be computed as the limit of starting with an arbitrary $Q_0^{(\pi)}$ and iteratively applying Equation (1). Compare these two processes. At the start, $Q_0^{(\pi)} = \overline{Q}_0^{(\pi)}$. After one iteration, $Q_1^{(\pi)} - \lambda \leq \overline{Q}_1^{(\pi)} \leq Q_1^{(\pi)}$. After a second iteration, $Q_2^{(\pi)} - \lambda(1 + \gamma) \leq \overline{Q}_2^{(\pi')} \leq Q_2^{(\pi)}$. Taking the limit gives the result. □

Let $V^{(\pi)}(s) = Q^{(\pi)}(s, \pi(s))$ and $\overline{V}^{(\pi)}(s) = \overline{Q}^{(\pi)}(s, \pi(s))$. Then Theorem 1 equivalently shows that $V^{(\pi)} - \lambda/(1 - \gamma) \leq \overline{V}^{(\pi)} \leq V^{(\pi)}$. Furthermore, the right inequality is an equality if and only if $\pi$ is the expert’s policy. This has two key consequences. First, the optimal policy $\pi^*$ for $\overline{M}$ is never worse than the expert’s policy $\pi^e$ when applied to $M$:

$$V^{(\pi^*)} \leq \overline{V}^{(\pi^*)} \leq V^{(\pi^*)} = V^{(\pi^e)}$$

Second, let $\pi^*$ be an optimal policy for $M$. Then

$$V^{(\pi^*)} \geq \overline{V}^{(\pi^*)} \geq \overline{V}^{(\pi^*)} \geq V^{(\pi^*)} - \lambda/(1 - \gamma)$$

Therefore, within an additive $\lambda/(1 - \gamma)$, an optimal policy for $\overline{M}$ is approximately optimal for $M$. In the worst case, this bound is tight. Consider an MDP $M$ with a a single state and two actions. The first has reward $\lambda$, the second has reward 0. The expert’s policy is to
take the second. Then for the $\overline{M}$, the expert’s policy is optimal, but it has value 0 for $\mathcal{M}$. Meanwhile, the optimal policy for $\mathcal{M}$ has value $\lambda/(1 - \gamma)$.

Thus, we have shown for our simplified version of DQfD that the suboptimality of the final policy is at most $\lambda/(1 - \gamma)$ regardless of the choice of expert. If the expert is particularly poor, this may lead to a policy better than the expert’s policy. However, if the gap between the quality of the expert’s policy and the optimal policy is smaller than this quality, learning may get stuck at a suboptimal policy like the expert’s. While this can be mitigated by shrinking $\lambda$, we have already seen that doing so can lead to poor learning in practice.

Appendix B. Online training

During pretraining, there is no access to the environment and the demonstration data is the only data available; during online training there is a second replay memory containing a fixed size set of the algorithm’s most recent observations. Because the large margin loss only applies to the demonstration data, controlling the ratio between demonstration data and online experience data has a similar effect to our simple heuristic of adapting the large margin weight. For example, if only half of the data is from the demonstrator, then the value of $\lambda$ is effectively cut in half. Notably, DQfD uses a loss-based prioritization scheme to weight the data, so it is possible that this could implicitly adapt $\lambda$ during online training, making the concerns discussed in the previous section less relevant in this setting. (This automatic adaptation is described as one of the contributions of DQfD.) Unfortunately, in practice this does not seem to occur. Figure 6a shows that with our bad demonstrator DQfD gets stuck at a low performance level whether we use loss-based prioritization or artificially impose a fixed ratio that half the data comes from demonstration and half from online experience. In contrast, plain RL or DQfD with a variant of our previous heuristic (where now the clipped ratio between learner performance and demonstrator performance determines the fraction of observations drawn from demonstration data) learning continues. Figure 6b shows that while loss-based prioritization does have an effect on the demonstration ratio, it tends to draw approximately thirty percent of its observations from the demonstration data after an initial period where the fraction is higher. In contrast, the adaptive demonstration ratio rapidly drives toward zero. Figures 6c and 6d show that with a good demonstrator, adapting the demonstration ratio is not particularly important (as was the case with adapting $\lambda$ in the offline setting, while with a bad demonstrator adapting the demonstration ratio alone suffices (there does not seem to be an additional benefit from combining our two heuristics).
While our results show that the loss-based prioritization does not actually have a meaningful effect on the balance between demonstration data and online experience data used by the algorithm, we do not rule out the possibility that it is still providing a benefit by making better decisions about which datapoints to sample from each set when drawing a minibatch.

Appendix C. Results on 32 Atari games

Table 1 shows our TD only and TD + LM’s results compared with the original DQfD on 32 Atari games. We exclude games that are not included in OpenAI Gym, and games where DoubleDQN performs too poorly to be meaningfully used as “expert demonstrator”. The field “Expert” corresponds to the score the demonstrator achieves when generating the demonstration data. For our TD only and TD + LM approach, demonstrations are gathered with an exploration rate of 0.1 and the size of the demonstration memory is 500K. For each game, we use around 20 different demonstrators and report the best learner and the corresponding quality of the demonstrator. The setting is identical to what has been used in the experiments reported in the main text. While it is not the purpose of our experiments, it is interesting to note that our results are consistent with the idea that “kickstarting” training by training an agent based on another agent does improve training speed and final performance (Schmitt et al., 2018).
<table>
<thead>
<tr>
<th></th>
<th>Hester et al. (2018)</th>
<th>TD only</th>
<th>TD + Large margin</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Expert</td>
<td>Size</td>
<td>Score</td>
</tr>
<tr>
<td>Alien</td>
<td>19425</td>
<td>19133</td>
<td>474</td>
</tr>
<tr>
<td>Amidar</td>
<td>1847</td>
<td>16790</td>
<td>176</td>
</tr>
<tr>
<td>Assault</td>
<td>1721</td>
<td>13224</td>
<td>634</td>
</tr>
<tr>
<td>Asterix</td>
<td>11300</td>
<td>9525</td>
<td>280</td>
</tr>
<tr>
<td>Atlantis</td>
<td>16350</td>
<td>17516</td>
<td>12737</td>
</tr>
<tr>
<td>Hank Heist</td>
<td>4182.5</td>
<td>32389</td>
<td>95</td>
</tr>
<tr>
<td>Battle Zone</td>
<td>47500</td>
<td>9072</td>
<td>14402</td>
</tr>
<tr>
<td>Beam Rider</td>
<td>16219</td>
<td>38665</td>
<td>366</td>
</tr>
<tr>
<td>Bowling</td>
<td>119</td>
<td>9991</td>
<td>93</td>
</tr>
<tr>
<td>Boxing</td>
<td>8</td>
<td>8438</td>
<td>8</td>
</tr>
<tr>
<td>Breakout</td>
<td>48</td>
<td>10475</td>
<td>4</td>
</tr>
<tr>
<td>Chopper Command</td>
<td>8000</td>
<td>7710</td>
<td>2486</td>
</tr>
<tr>
<td>Crazy Climber</td>
<td>46100</td>
<td>18937</td>
<td>14051</td>
</tr>
<tr>
<td>Demon Attack</td>
<td>3995</td>
<td>17409</td>
<td>148</td>
</tr>
<tr>
<td>Fishing Derby</td>
<td>5</td>
<td>6388</td>
<td>-74</td>
</tr>
<tr>
<td>Freeway</td>
<td>31</td>
<td>10239</td>
<td>23</td>
</tr>
<tr>
<td>Gopher</td>
<td>12510</td>
<td>38632</td>
<td>1142</td>
</tr>
<tr>
<td>Gravitar</td>
<td>8175</td>
<td>15377</td>
<td>278</td>
</tr>
<tr>
<td>James Bond</td>
<td>525</td>
<td>9050</td>
<td>262</td>
</tr>
<tr>
<td>Kangaroo</td>
<td>24350</td>
<td>20984</td>
<td>917</td>
</tr>
<tr>
<td>Krull</td>
<td>10885</td>
<td>32581</td>
<td>2217</td>
</tr>
<tr>
<td>Kung Fu Master</td>
<td>17110</td>
<td>12989</td>
<td>557</td>
</tr>
<tr>
<td>MsPacman</td>
<td>43401</td>
<td>21896</td>
<td>692</td>
</tr>
<tr>
<td>Name This Game</td>
<td>15365</td>
<td>43571</td>
<td>3745</td>
</tr>
<tr>
<td>Pong</td>
<td>-6</td>
<td>17719</td>
<td>-20</td>
</tr>
<tr>
<td>Q-bert</td>
<td>90075</td>
<td>75472</td>
<td>5134</td>
</tr>
<tr>
<td>River Raid</td>
<td>28475</td>
<td>46233</td>
<td>2148.5</td>
</tr>
<tr>
<td>Road Runner</td>
<td>14300</td>
<td>5574</td>
<td>8795</td>
</tr>
<tr>
<td>Seaquest</td>
<td>78815</td>
<td>57453</td>
<td>196</td>
</tr>
<tr>
<td>Np N Down</td>
<td>11330</td>
<td>10421</td>
<td>1817</td>
</tr>
<tr>
<td>Video Pinball</td>
<td>20415</td>
<td>10051</td>
<td>10656</td>
</tr>
<tr>
<td>Yars’ Revenge</td>
<td>65942</td>
<td>21334</td>
<td>4226</td>
</tr>
</tbody>
</table>

Table 1: Our DQfD compared with the original DQfD (with different demonstration data). The number of training steps is 750K for Hester et al. (2018) and 4M for ours.
Figure 7: TD only: relation between the quality of demonstration data and learner score.
Figure 8: TD + LM: relation between the quality of demonstration data and learner score.