Learning from Demonstrations for Real World Reinforcement Learning

Todd Hester TODDHESTER@GOOGLE.COM
Matej Vecerik MATEJVECERIK@GOOGLE.COM
Olivier Pietquin PIETQUIN@GOOGLE.COM
Marc Lanctot LANCTOT@GOOGLE.COM
Tom Schaul SCHAUL@GOOGLE.COM
Bilal Piot PIOT@GOOGLE.COM
Andrew Sendonaris SENDOS@GOOGLE.COM
Gabriel Dulac-Arnold GABE@SQUIRRELSOUP.NET
Ian Osband IOSBAND@GOOGLE.COM
John Agapiou JAGAPIOU@GOOGLE.COM
Joel Z. Leibo JZL@GOOGLE.COM
Audrunas Gruslys AUDRUNAS@GOOGLE.COM
Google DeepMind, 6 Pancras Square, London, UK N1C 4AG

Abstract

Deep reinforcement learning (RL) has achieved several high profile successes in difficult control problems. However, these algorithms typically require a huge amount of data before they reach reasonable performance. In fact, their performance during learning can be extremely poor. This may be acceptable for a simulator, but it severely limits the applicability of deep RL to many real-world tasks, where the agent must learn in the real environment. In this paper we study a setting where the agent may access data from previous control of the system. We present an algorithm, *Deep Q-learning from Demonstrations* (DQfD), that leverages this data to massively accelerate the learning process even from relatively small amounts of demonstration data. DQfD works by combining temporal difference updates with large-margin classification of the demonstrator's actions. We show that DQfD has better initial performance than Deep Q-Networks (DQN) on 40 of 42 Atari games and it receives more average rewards than DQN on 27 of 42 Atari games. We also demonstrate that DQfD learns faster than DQN even when given poor demonstration data.

1. Introduction

Over the past few years, there have been a number of successes in learning policies for sequential decision-making problems and control. Notable examples include deep model-free Q-learning for general Atari game-playing (Mnih et al., 2015), end-to-end policy search for control of robot motors (Levine et al., 2016), model predictive control with embeddings (Watter et al., 2015), and strategic policies that combined with search led to defeating a top human expert at the game of Go (Silver et al., 2016). An important part of the success of these approaches has been to leverage the recent contributions to scalability and performance of deep learning (LeCun et al., 2015). The approach taken in (Mnih et al., 2015) builds a data set of previous experience using batch RL to train large convolutional neural networks in a supervised fashion from this data. As a result, the correlation in labels or values from state distribution bias is mitigated, leading to good (in many cases, super-human) control policies.

It still remains difficult to apply these algorithms to real world settings such as data centers, autonomous vehicles (Hester & Stone, 2013), helicopters (Abbeel et al., 2007), or recommendation systems (Shani et al., 2005). Typically these algorithms learn good control policies only after many millions of steps of very poor performance in simulation. This situation is acceptable when there is a perfectly accurate simulator; however, many real world problems do not come with such a simulator. Instead, in these situations, the agent must learn in the real domain with real consequences for its actions, which requires that the agent have good on-line performance from the start of learning. While accurate simulators are difficult to find, most of these problems have data of the system operating under a previous controller (either human or machine) that performs reasonably well. In this work, we make use of this demonstration data to pre-train the agent so that it can perform well

in the task from the start of learning, and then continue improving from its own self-generated data. Enabling learning in this framework opens up the possibility of applying RL to many real world problems where demonstration data is common but simulators do not exist.

We propose a new deep reinforcement learning algorithm, *Deep Q-learning from Demonstrations* (DQfD), which leverages demonstration data to massively accelerate learning. DQfD initially pre-trains solely on the demonstration data using a combination of temporal difference (TD) and supervised losses. The supervised loss enables the algorithm to learn to imitate the demonstrator while the TD loss enables it to learn a valid value function from which it can continue learning with RL. After pre-training, the agent starts interacting with the domain with its learned policy. The agent keeps the demonstration data and its new self-generated data in separate replay buffers and each mini-batch it uses to update its network has a set proportion of data from each buffer. This algorithm out-performs pure reinforcement learning using Double DQN (van Hasselt et al., 2016) in average rewards on 27 of 42 Atari games, and out-performs pure imitation learning on 31 of 42 Atari games (Bellemare et al., 2013). DQfD learns to out-perform the demonstrator on six games. Finally, in a toy domain and one selected Atari game, DQfD receives more average rewards than both DQN and imitation *despite* being given bad demonstration data.

2. Background

We adopt the standard Markov Decision Process (MDP) formalism for this work (Sutton & Barto, 1998). An MDP is defined by a tuple $\langle S, A, R, T, \gamma \rangle$, which consists of a set of states S, a set of actions A, a reward function $R(s,a)$, a transition function $T(s,a,s') = P(s'|s,a)$, and a discount factor γ. In each state $s \in S$, the agent takes an action $a \in A$. Upon taking this action, the agent receives a reward $R(s,a)$ and reaches a new state s', determined from the probability distribution $P(s'|s,a)$. A policy π specifies for each state which action the agent will take.

The goal of the agent is to find the policy π mapping states to actions that maximizes the expected discounted total reward over the agent's lifetime. The value $Q^\pi(s,a)$ of a given state-action pair (s,a) is an estimate of the expected future reward that can be obtained from (s,a) when following policy π. The optimal value function $Q^*(s,a)$ provides maximal values in all states and is determined by solving the Bellman equation:

$$Q^*(s,a) = \mathbb{E}\left[R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q^*(s',a')\right]. \tag{1}$$

The optimal policy π is then:

$$\pi(s) = \operatorname*{argmax}_a Q^*(s,a). \tag{2}$$

DQN (Mnih et al., 2015) approximates the value function $Q(s,a)$ with a deep neural network that outputs a set of action values $Q(s,\cdot;\theta)$ for a given state input s, where θ are the parameters of the network. There are two key components of DQN that make this work. First, it uses a separate target network that is copied every τ steps from the regular network so that the target Q-values are more stable. Second, the agent adds all of its experiences to a replay buffer $\mathcal{D}^{replay}$, which is then sampled uniformly to perform updates on the network.

Double DQN (van Hasselt et al., 2016) adds double Q-learning, where the current network is used to calculate the argmax over next state values and the target network is used to get the value of that action. The learning target becomes:

$$J_{DQ}(Q) = \left(R(s,a) + \gamma Q(s_{t+1}, a^{\max}_{t+1}; \theta') - Q(s,a;\theta)\right)^2, \tag{3}$$

where θ' are the parameters of the target network, and $a^{\max}_{t+1} = \operatorname{argmax}_a Q(s_{t+1}, a; \theta)$. Separating the value functions used for these two variables reduces the upward bias that is created with regular Q-learning updates, enabling RL-compatible pre-training.

3. Deep Q-Learning from Demonstrations

In many real-world settings of reinforcement learning, we have access to data of the system being operated by its previous controller, but we do not have access to an accurate simulator of the system. Therefore, we want the agent to learn as much as possible from the demonstration data before running on the real system. The goal of the pre-training phase is to learn to imitate the demonstrator with a value function that satisfies the Bellman equation so that it can be updated with TD updates once the agent starts interacting with the environment. During this pre-training phase, the agent samples mini-batches from the demonstration data and updates the network by applying three losses: the double Q-learning loss, a supervised large margin classification loss, and an L2 regularization loss on the network weights and biases. The supervised loss is used for classification of the demonstrator's actions, while the Q-learning loss ensures that the network satisfies the Bellman equation and can be used as a starting point for TD learning.

The supervised loss is critical for the pre-training to have any effect. Since the demonstration data is necessarily covering a narrow part of the state space and not taking all possible actions, many state-actions have never been taken and have no data to ground them to realistic values. If we were to pre-train the network with only Q-learning updates

towards the max value of the next state, the network would update towards the highest of these ungrounded variables and the network would propagate these values throughout the Q function. Adding the large margin classification loss grounds the values of the unseen actions to reasonable values, and makes the greedy policy induced by the value function imitate the demonstrator (Piot et al., 2014a):

$$J_E(Q) = \max_{a \in A}[Q(s,a) + l(s,a_E,a)] - Q(s,a_E) \quad (4)$$

where a_E is the action the expert demonstrator took in state s and $l(s,a_E,a)$ is a margin function that is 0 when $a = a_E$ and positive otherwise. This loss forces the values of the other actions to be at least a margin lower than the value of the demonstrator's action. If the algorithm pre-trained with only this supervised loss, there would be nothing constraining the values between consecutive states and the Q-network would not satisfy the Bellman equation, which is required to improve the policy on-line with TD learning.

We also add an L2 regularization loss applied to the weights and biases of the network to help prevent it from over-fitting on the relatively small demonstration dataset. The overall loss used to update the network is a combination of all three losses:

$$J(Q) = J_{DQ}(Q) + \lambda_1 J_E(Q) + \lambda_2 J_{L2}(Q). \quad (5)$$

The λ parameters control the weighting between the losses.

Once the pre-training phase is completed, ideally the agent will have learned a reasonable policy that is safe to run on the real system. In the next phase, the agent starts acting on the system, collecting self-generated data, and adding it to its agent replay buffer $\mathcal{D}^{replay}$. Data is added to the agent replay buffer until it is full, and then the agent starts over-writing old data in that buffer. Meanwhile the demonstration data is still maintained in a separate demonstration replay buffer $\mathcal{D}^{demo}$, which stays constant. Each mini-batch contains n samples with the portion of demonstration data defined by parameter $p = \frac{n^{demo}}{n^{demo}+n^{replay}}$. For the self-generated data, only the double Q-learning loss is applied, while for the demonstration data, both the supervised and double Q-learning losses are applied.

Overall, Deep Q-learning from Demonstration (DQfD) differs from DQN in five key ways (examined in Section 4.2.2):

- Pre-training: DQfD initially trains solely on the demonstration data before starting any interaction with the environment. Pre-training happens with a combination of Q-learning loss and supervised loss so that the agent imitates the demonstrator while having a value function ready for TD learning.
- Supervised losses: In addition to TD losses, a large margin supervised loss is applied that pushes the value of the demonstrator's actions above the other action values (Piot et al., 2014a).
- L2 Regularization losses: The algorithm also adds L2 regularization losses on the network weights to prevent over-fitting on the demonstration data.
- Separate datasets: Demonstration data is stored in $\mathcal{D}^{demo}$ and never overwritten, while self-generated data is stored in $\mathcal{D}^{replay}$ and overwritten as usual.
- Controlled data sampling: The proportion of demonstration data versus self-generated data is controlled in each mini-batch, with $p = \frac{n^{demo}}{n^{demo}+n^{replay}}$.

Pseudo-code is sketched in Algorithm 1. The behavior policy is an ϵ-greedy policy with respect to the Q_θ values.

Algorithm 1 Deep Q-learning from Demonstrations.

Inputs: $\mathcal{D}^{demo}$: demonstration data set, $\mathcal{D}^{replay}$: empty, θ: weights for initial behavior network (random), θ': weights for target network (random), τ: frequency at which to update target net, k: number of pre-training gradient updates
for steps $t \in \{1, 2, \ldots k\}$ **do**
  Sample a mini-batch of n transitions from $\mathcal{D}^{demo}$
  Calculate loss $J(Q)$ using target network (Eq. 5)
  Perform a gradient descent step to update θ
end for
for steps $t \in \{1, 2, \ldots\}$ **do**
  Sample action from behavior policy $a \sim \pi^{\epsilon Q_\theta}$
  Play action a and observe (s', r).
  Store tuple (s, a, r, s') into $\mathcal{D}^{replay}$, overwriting oldest if over capacity
  Sample a mini-batch of n transitions from $\mathcal{D}^{demo} \cup \mathcal{D}^{replay}$ with a fraction p of the samples from $\mathcal{D}^{demo}$
  Calculate loss $J(Q)$ using target network (Eq. 5)
  Perform a gradient descent step to update θ
  if $t \bmod \tau = 0$ **then** $\theta' \leftarrow \theta$ **end if**
  $s \leftarrow s'$
end for

4. Experimental Results

For all of our experiments, we evaluated three different algorithms, each averaged across four trials:

- Full DQfD algorithm
- Double DQN learning without any demonstration data
- Supervised imitation from demonstration data without any environment interaction

For DQfD, we initially performed informal parameter tuning on four Atari games (Bellemare et al., 2013). DQfD was run with the following parameters:

- Pre-training with 1,000,000 mini-batch updates.

- Expert Sampling Ratio $p = 0.1$.
- Supervised loss weight $\lambda_1 = 1.0$.
- L2 regularization weight $\lambda_2 = 10^{-5}$.
- Expert margin $l(s, a_E, a) = 0.8$ when $a \neq a_E$.
- ϵ-greedy exploration with $\epsilon = 0.01$, which is the same used by Double DQN (van Hasselt et al., 2016).

Double DQN was run with the same exploration and L2 regularization (λ_2) as DQfD, but no pre-training, no expert sampling, and no supervised loss.

For the supervised imitation comparison, we performed supervised classification of the demonstrator's actions using a cross-entropy loss, with the same network architecture and L2 regularization used by DQfD and DQN. The imitation algorithm did not use any TD loss.

4.1. Catch

We first evaluated the agent on a simple domain called Catch where it is easy to generate optimal demonstration data. In this domain, there is a falling ball and the agent must move across the bottom of the screen to catch the ball. The state is represented by a screen of 25x10 pixels, each valued 0 or 1. Only two pixels will be 1, the ball and the agent. At the start of each episode, the ball is in the top row in a random column, and the agent is in the bottom row in the center column. The ball falls one row each step. When the ball reaches the bottom row, the episode terminates with reward +1 if the agent is in the same column as the ball, and a reward of -1 otherwise. The agent has 10 actions: move left, move right, and eight actions that do nothing. Each time step, there is a 10% chance that the agent will move left regardless of the action it takes. The added stochasticity requires generalization of the demonstration data from the transitions that were seen, and the extra no-op actions make exploration in this task more difficult. As it only takes 10 steps for the ball to fall down, some columns the ball appears in are unreachable as they are more than 10 steps away from the center column where the agent starts. All three algorithms use a feed-forward network with one layer of 50 hidden units.

We generated demonstration data by training DQN on the task until it learned an optimal policy and then generating 1,000 transitions from this policy. Figure 1 shows results learning from this optimal demonstration, with 200 iterations of 250 steps each. DQfD starts out at similar performance to pure imitation learning and then improves from there. Meanwhile, DQN starts out at random performance and improves from there.

When we switch from using optimal demonstrations to demonstration data with 10% random actions, the performance of imitation learning drops much more than the performance of DQfD. DQfD is able to generalize the poorer demonstration data better using its three losses, and starts at similar initial performance with both datasets. As data collected from real-world experiments often comes from noisy sensor readings or unreliable human laborers, it is important for learning algorithms to be robust to imperfections in the demonstrations.

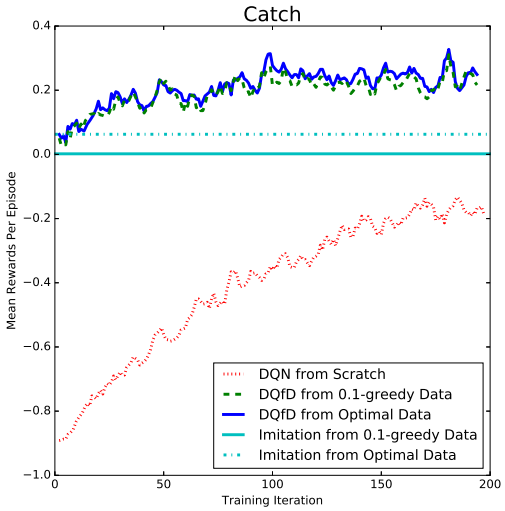

Figure 1. On-line rewards of all three algorithms on the game of Catch, when given either optimal or ϵ-greedy demonstrations of 1,000 transitions. DQfD is able to perform similarly well even when the demonstration data has 10% random actions.

4.2. Atari

We next evaluated DQfD on a much more challenging domain, the Arcade Learning Environment (ALE) (Bellemare et al., 2013). ALE is a set of Atari games that are a standard benchmark for DQN and contains many games on which humans still perform better than the best learning agents. The agent plays the Atari games from a down-sampled 84x84 image of the game screen that has been converted to greyscale, and the agent stacks four of these frames together as its state. The agent must output one of 18 possible actions for each game. The agent applies a discount factor of 0.99 and all of its actions are repeated for four Atari frames. We use the same convolutional network architecture used by DQN (Mnih et al., 2015).

We ran experiments on a subset of 42 Atari games. We had a human player play each game between three and twelve times. Each episode was played either until the game terminated or for 20 minutes. During game play, we logged the agent's state, actions, rewards, and terminations. Table 1 shows the total number of transitions collected from the human demonstrator in each game, which ranges from 5574 to 75472 transitions per game. DQfD learns from a very small dataset compared to other similar work, as AlphaGo (Silver et al., 2016) learns from 30 million human

Game	Demonstrator Worst Score	Demonstrator Best Score	Number Transitions	Number Episodes
Alien	9690	29160	19133	5
Amidar	1353	2341	16790	5
Assault	1168	2274	13224	5
Asterix	4500	18100	9525	5
Asteroids	14170	18100	22801	5
Atlantis	10300	22400	17516	12
Bank Heist	900	7465	32389	7
Battle Zone	35000	60000	9075	5
Beam Rider	12594	19844	38665	4
Bowling	89	149	9991	5
Boxing	0	15	8438	5
Breakout	17	79	10475	9
Chopper Command	4700	11300	7710	5
Crazy Climber	30600	61600	18937	5
Defender	5150	18700	6421	5
Demon Attack	1800	6190	17409	5
Double Dunk	-22	-14	11855	5
Enduro	383	803	42058	5
Fishing Derby	-10	20	6388	4
Freeway	30	32	10239	5
Gopher	2500	22520	38632	5
Gravitar	2950	13400	15377	5
Hero	35155	99320	32907	5
Ice Hockey	-4	1	17585	5
James Bond	400	650	9050	5
Kangaroo	12400	36300	20984	5
Krull	8040	13730	32581	5
Kung Fu Master	8300	25920	12989	5
Montezuma's Revenge	32300	34900	17949	5
Ms Pacman	31781	55021	21896	3
Name This Game	11350	19380	43571	5
Pitfall	3662	47821	35347	5
Pong	-12	0	17719	3
Private Eye	70375	74456	10899	5
Q-bert	80700	99450	75472	5
River Raid	17240	39710	46233	5
Road Runner	8400	20200	5574	5
Seaquest	56510	101120	57453	7
Solaris	2840	17840	28552	6
Up N Down	6580	16080	10421	4
Video Pinball	8409	32420	10051	5
Yars' Revenge	48361	83523	21334	4

Table 1. Atari games the algorithm was evaluated on along with the best and worst scores the human demonstrator achieved on the game, and the number of trials and transitions collected.

Game	DQfD	Double DQN	Imitation
Alien	**577.1**	280.1	473.9
Amidar	**250.4**	76.3	175.0
Assault	1017.4	**1384.9**	634.4
Asterix	2353.3	**4715.4**	279.9
Asteroids	**2507.8**	914.6	1267.3
Atlantis	**17647.0**	13494.8	12736.6
Bank Heist	**106.2**	8.7	95.2
Battle Zone	12486.1	3456.4	**14402.4**
Beam Rider	464.3	**748.8**	365.9
Bowling	46.5	28.5	**92.6**
Boxing	**89.1**	85.2	7.5
Breakout	**95.8**	5.3	3.5
Chopper Command	**2989.3**	2582.3	2485.7
Crazy Climber	103980.9	**108450.5**	14051.0
Defender	**7607.6**	3505.7	3819.1
Demon Attack	186.0	**405.1**	147.5
Double Dunk	**-16.9**	-20.2	-21.4
Enduro	624.8	**736.8**	134.8
Fishing Derby	-18.0	**-13.5**	-74.4
Freeway	**30.9**	28.5	22.7
Gopher	**9079.5**	4909.8	1142.6
Gravitar	245.1	35.6	**248.0**
Hero	**20428.2**	5373.0	5903.3
Ice Hockey	-9.8	**-4.7**	-13.5
James Bond	145.5	6.5	**262.1**
Kangaroo	1311.2	**1779.2**	917.3
Krull	1054.8	1880.2	**2216.6**
Kung Fu Master	**12328.6**	6677.8	556.7
Montezuma's Revenge	**780.9**	0.0	576.3
Ms Pacman	680.6	308.8	**692.4**
Name This Game	**4376.5**	4171.5	3745.3
Pitfall	-124.6	-26.3	**182.8**
Pong	**15.2**	13.6	-20.4
Private Eye	38280.5	-111.3	**42749.6**
Q-bert	2211.6	245.9	**5133.8**
River Raid	2368.0	**3202.6**	2148.5
Road Runner	38041.5	**39988.2**	8794.9
Seaquest	181.2	**1113.9**	195.6
Solaris	3107.9	221.8	**3589.6**
Up N Down	**10265.1**	8522.9	1816.7
Video Pinball	**10926.2**	7135.5	10655.5
Yars' Revenge	4764.3	**5731.8**	4225.8

Table 2. Average On-line Rewards of each algorithm over 200 iterations of 1 million Atari frames each on all 42 Atari games.

transitions, and DQN (Mnih et al., 2015) learns from over 50 million frames. DQfD's smaller demonstration dataset makes it more difficult to learn a good representation without over-fitting. Table 1 lists the games we selected as well as the demonstrator's best and worst performance on each game. Our human demonstrator is much better than DQN on some games (e.g. Private Eye, Pitfall), but much worse than DQN on many games (e.g. Breakout, Pong).

We found that in many of the games where the human player is better than DQN, it was due to DQN being trained with all rewards clipped to 1.0 (Mnih et al., 2015). For example, in Private Eye, DQN has no reason to go for actions that reward +25,000 versus actions that reward +10. To make the reward function used by the human demonstrator and the agent more consistent, we had the agent scale the rewards to have maximum value 1.0 by dividing by the maximal absolute reward it has seen. The agent tracks the maximum reward it has seen (in demonstration or self-generated data) over time and re-scales the rewards as it samples them from the replay buffer. For DQfD, the highest one-step reward usually exists in the demonstration data set, and the rewards are scaled appropriately from the start. For DQN, this will make the rewards non-stationary. (van Hasselt et al., 2016) perform a similar adaptive re-scaling of targets and show that the change causes DQN to improve on some games and perform worse on others. Overall, this reward scaling makes both algorithms use the same true reward function that the demonstrator uses.

4.2.1. Main Results

In real world tasks, the agent must perform well from its very first action. Therefore, we evaluate the agent on average on-line rewards, rather than just looking at the value of its final policy. Table 2 shows the average rewards achieved by each algorithm in every game over 200 iterations of one million Atari frames each. DQfD outperforms Double DQN in average rewards on 27 of the 42 games, and outperforms imitation learning on 31 of the 42 games.

Figure 2 shows results on Hero, which was typical of many of the games that were run. Plots showing the results across all 42 games are included in the Appendix. DQfD starts out

Figure 2. On-line rewards of the three algorithms on the game of Hero. Many of the games had similar results to this one, where DQfD starts out with performance near the imitation policy and improves from there.

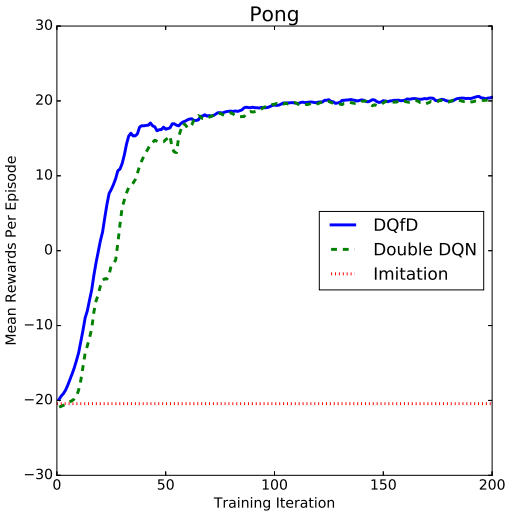

Figure 3. On-line rewards of the three algorithms on the game of Pong. Although the human demonstrator did not win a game in their demonstrations, DQfD still out-performs DQN for the first 58 iterations.

with performance near that of the imitation policy, and continues to improve from there. Meanwhile, DQN starts out at random performance and slowly improves from there. Imitation learning is always a flat line because it will not improve with interactions in the environment.

One of the key components of DQfD is pre-training the agent so it can perform reasonably well from its very first action, which is critical for real world tasks. DQfD starts out with better performance than DQN on the very first iteration on all but two games. In addition, on 23 games, DQfD starts out with higher performance than pure imitation learning, as the addition of the TD loss helps the agent generalize the demonstration data better. In (Piot et al., 2014b), it was shown that adding a TD loss improved imitation performance even without any rewards in the domain.

DQfD learns to out-perform the worst demonstration episode it was pre-trained on in 15 games and it learns to play better than the best demonstration episode in six of the games: Boxing, Breakout, Crazy Climber, Pong, Road Runner, and Up N Down. Pong, shown in Figure 3, is a particularly interesting case as DQfD performs better than DQN on the first 58 iterations even though the demonstration data it was pre-trained on was poor (the demonstrator did not win a single game). DQfD converges to a better final policy than DQN on 24 of the 42 games.

On some of the more difficult games such as Pitfall, DQfD actually gets worse when it starts interacting with the game. In many of these games, the Atari reward function with a discount factor of 0.99 is malformed (e.g. in Pitfall the first positive reward is seven screens away and will be discounted close to 0). In all of the games, pure imitation learning is worse than the demonstrator's performance, and in most games imitation learning is not able to classify the expert's action perfectly even on the states in the demonstration dataset.

4.2.2. DQfD Ablation Studies

Next, we looked at the impact of each of the five major differences between DQfD and DQN. There are quite a few games where DQN learns quite quickly, and DQfD gets a boost in initial performance and still learns as fast as DQN (e.g. Boxing, Pong, Freeway). We investigate the impact of the expert sampling ratio p on one of these games, Freeway, in Figure 4. As the sampling ratio is decreased and the agent sees more self-generated game interactions, it learns more quickly.

Road Runner (Figure 5) is another interesting game, where DQN learns a score exploit which is much different from how a human would play the game. DQfD starts out with better initial performance than DQN and continues learning from there, surpassing the best performance of the human in demonstrations, but not matching DQN's final performance. We examined the impact of pre-training on Road Runner. The agent with pre-training receives more rewards on the first iterations, showing the clear advantage of pre-training.

Figure 6 shows comparisons of the algorithm with each of the three losses applied to demonstration data removed, on the game of Private Eye. The removal of the TD loss in pre-training has an impact initially that goes away as the agent turns its action classification into a value function. The regularization loss impacts the starting performance

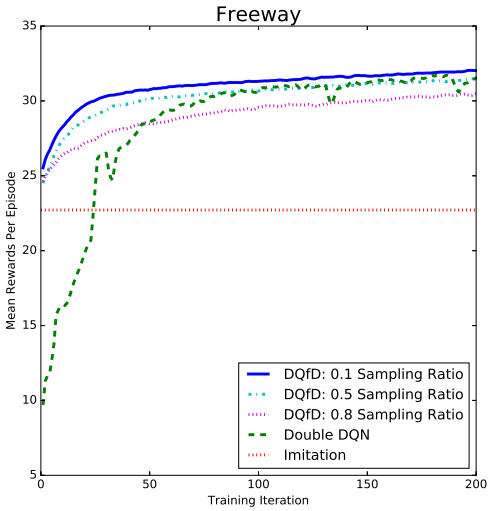

Figure 4. On-line rewards of DQfD with various expert sampling ratios, on the game of Freeway.

Figure 5. On-line rewards of DQfD with and without pre-training on the game of Road Runner. Pre-training gives Road Runner an early boost in performance, and DQfD out-performs DQN until DQN learns a score exploit.

of the agent and the policy it converges to. As expected, pre-training without any supervised loss results in a network trained towards ungrounded Q-learning targets and the agent is unable to recover from this poorly trained network. These results are representative of the results across the full set of games. With all three losses, DQfD learns a policy better than the best published results on this game.

Figure 6 also compares using a large margin and cross entropy loss for the classification of the demonstrator's actions. (Lakshminarayanan et al., 2016) use a cross entropy loss in their approach, but Figure 6 shows that it results in worse performance than using the large margin loss. This difference is likely because the cross-entropy loss is less compatible with the Q-learning loss as it pushes the action values as far apart as possible.

5. Related Work

Imitation learning is primarily concerned with matching the performance of the demonstrator. One popular algorithm, DAGGER (Ross et al., 2011), iteratively produces new policies based on polling the expert policy outside its original state space, showing that this leads to no-regret over validation data in the online learning sense. DAGGER requires the expert to be available during training to provide additional feedback to the agent. Another popular paradigm is to setup a zero-sum game where the learner chooses a policy and the adversary chooses a reward function (Syed & Schapire, 2007; Syed et al., 2008; Ho & Ermon, 2016). Demonstrations have also been used for inverse optimal control in high-dimensional, continuous robotic control problems (Finn et al., 2016). However, these approaches only do imitation learning and do not allow for learning from task rewards.

Recently, demonstration data has been shown to help in difficult exploration problems in RL (Subramanian et al., 2016). There has also been recent interest in this combined imitation and RL problem. For example, the HAT algorithm transfers knowledge directly from human policies (Taylor et al., 2011). Follow-ups to this work showed how expert advice or demonstrations can be used to shape rewards in the RL problem (Brys et al., 2015; Suay et al., 2016). A different approach is to shape the policy that is used to sample experience (Cederborg et al., 2015), or to use policy iteration from demonstrations (Kim et al., 2013; Chemali & Lezaric, 2015).

Our algorithm works in a scenario where rewards are given by the environment used by the demonstrator. This framework was appropriately called Reinforcement Learning with Expert Demonstrations (RLED) in (Piot et al., 2014a) and is also evaluated in (Kim et al., 2013; Chemali & Lezaric, 2015). Our setup is similar to (Piot et al., 2014a) in that we combine TD and classification losses in a batch algorithm in a model-free setting; ours differs in that our agent is pre-trained on the demonstration data initially and the batch of self-generated data grows over time and is used as experience replay to train deep Q-networks. (Piot et al., 2014b) present interesting results showing that adding a TD loss to the supervised classification loss improves imitation learning even when there are no rewards.

While our algorithm works on the RLED framework, we emphasize that the method we present is not restricted by this and could be combined with inverse RL methods (Ng & Russell, 2000; Abbeel & Ng, 2004;

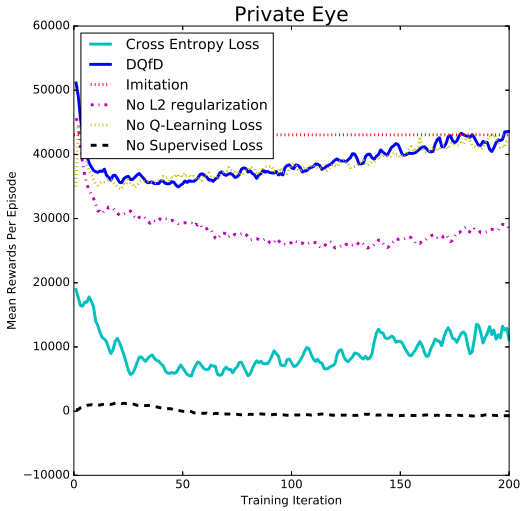

Figure 6. On-line rewards of DQfD with each demonstration data loss removed on the game of Private Eye (showing only the first 50 iterations). Removing any of the losses degrades the performance of the algorithm.

Babes-Vroman et al., 2011; Piot et al., 2013) to produce reward functions that are then used in place of task rewards.

Another work that is similarly motivated to ours is (Schaal, 1996). This work is focused on real world learning on robots, and thus is also concerned with on-line performance. Similar to our work, they pre-train the agent with demonstration data before letting it interact with the task. However, they do not use supervised learning to pre-train their algorithm, and are only able to find one case where pre-training helps learning on Cart-Pole.

AlphaGo (Silver et al., 2016) takes a similar approach to our work in pre-training from demonstration data before interacting with the real task. AlphaGo first trains a policy network from a dataset of 30 million expert actions, using supervised learning to predict the actions taken by experts. It then uses this as a starting point to apply policy gradient updates during self-play, combined with planning rollouts. Here, we do not have a model available for planning, so we focus on the model-free Q-learning case.

Human Experience Replay (Hosu & Rebedea, 2016) is an algorithm in which the agent samples from a replay buffer that is mixed between agent and demonstration data, similar to our approach. Gains were only slightly better than a random agent, and were surpassed by their alternative approach, Human Checkpoint Replay, which requires the ability to set the state of the environment. While their algorithm is similar in that it samples from two buffers, it does not pre-train the agent or use a supervised loss. Our results show higher scores over a larger variety of games, without requiring full access to the environment. (Lipton et al., 2016) show promising results with initializing the DQN agent's replay buffer with demonstration data on dialog tasks, but they do not pre-train the agent for good initial performance.

The work that most closely relates to ours is a workshop paper (Lakshminarayanan et al., 2016). They are also combining TD and classification losses in a deep Q-learning setup. They use a trained DQN agent to generate their demonstration data, which on most games is better than human data. It also guarantees that the policy used by the demonstrator can be represented by the apprenticeship agent as they are both using the same state input and network architecture. They use a cross-entropy classification loss rather than the large margin loss DQfD uses and they do not pre-train the agent to perform well from its first interactions with the environment. Our experiments in Section 4.2.2 show that both of these differences are crucial for the agent. In particular, the cross-entropy loss does not combine well with the double Q-learning loss.

6. Discussion

The learning framework that we have presented in this paper is one that is very common in real world problems such as controlling data centers, autonomous vehicles (Hester & Stone, 2013), or recommendation systems (Shani et al., 2005). In these problems, typically there is no accurate simulator available, and learning must be performed on the real system with real consequences. However, there is often data available of the system being operated by a previous controller. We have presented a new algorithm called DQfD that takes advantage of this data to accelerate learning on the real system. It first pre-trains solely on demonstration data, using a combination of TD and supervised losses so that it has a reasonable policy that is a good starting point for learning in the task. Once it starts interacting with the task, it continues learning by sampling from both its self-generated data as well as the demonstration data.

We have shown that this algorithm has better initial performance than DQN on 40 of 42 Atari games, and outperforms it in the average on-line rewards it receives on 27 of 42 Atari games. In addition, DQfD learns to perform better than its best demonstration episode on six of the games, and outperforms both DQN and imitation learning even when given intentionally poor demonstration data. DQfD's ability to perform well initially and continue learning from there enables RL on a wide range of real world systems for which approaches like DQN were not previously applicable because they had to learn from scratch.

These results may seem obvious given that DQfD has access to privileged data, but the rewards and demonstrations are mathematically dissimilar training signals, and naive

approaches to combining them can have disastrous results. We argue that the combination of all three losses during pre-training is critical for the agent to learn a single coherent representation that is not destroyed by the switch in training signals after pre-training.

There are many reasons why learning from human data is difficult. In most games, imitation learning is unable to perfectly classify the demonstrator's actions even on the demonstration dataset. The human demonstrator is playing the game in a way that is impossible for the agent to represent with its state observations. The human may also be executing a high rewarding policy that is much different from the policy DQN would learn. In future work, we plan to measure these differences between demonstration and agent data to inform approaches that derive more value from the demonstrations. Another future direction is to apply these concepts to domains with continuous actions, where the classification loss becomes a regression loss.

Acknowledgments

The authors would like to thank Keith Anderson, Chris Apps, Ben Coppin, Nando de Freitas, Chris Gamble, Thore Graepel, Georg Ostrovski, Cosmin Paduraru, Jack Rae, Amir Sadik, Jon Scholz, David Silver, Tom Stepleton, Ziyu Wang, and many others at DeepMind for insightful discussions, code contributions, and other efforts.

References

Abbeel, P. and Ng, A. Y. Apprenticeship learning via inverse reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2004.

Abbeel, P., Coates, A., Quigley, M., and Ng, A. Y. An application of reinforcement learning to aerobatic helicopter flight. In *Advances in Neural Information Processing Systems (NIPS)*, 2007.

Babes-Vroman, M., Marivate, V., Subramanian, K., and Littman, M. Apprenticeship learning about multiple intentions. In *International Conference on Machine Learning (ICML)*, 2011.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The arcade learning environment: An evaluation platform for general agents. *Journal of Artifificial Intelligence Research (JAIR)*, 47:253–279, 2013.

Brys, T., Harutyunyan, A., Suay, H.B., Chernova, S., Taylor, M.E., and Nowé, A. Reinforcement learning from demonstration through shaping. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2015.

Cederborg, T., Grover, I., Isbell, C.L., and Thomaz, A.L. Policy shaping with human teachers. In *International Joint Conference on Artificial Intelligence (IJCAI 2015)*, 2015.

Chemali, J. and Lezaric, A. Direct policy iteration from demonstrations. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2015.

Finn, C., Levine, S., and Abbeel, P. Guided cost learning: Deep inverse optimal control via policy optimization. In *International Conference on Machine Learing (ICML)*, 2016.

Hester, Todd and Stone, Peter. TEXPLORE: Real-time sample-efficient reinforcement learning for robots. *Machine Learning*, 90(3), 2013.

Ho, J. and Ermon, S. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

Hosu, I.-A. and Rebedea, T. Playing atari games with deep reinforcement learning and human checkpoint replay. In *ECAI Workshop on Evaluating General Purpose AI*, 2016.

Kim, B., Farahmand, A., Pineau, J., and Precup, D. Learning from limited demonstrations. In *Advances in Neural Information Processing Systems (NIPS)*, 2013.

Lakshminarayanan, Aravind S., Ozair, Sherjil, and Bengio, Yoshua. Reinforcement learning with few expert demonstrations. In *NIPS Workshop on Deep Learning for Action and Interaction*, 2016.

LeCun, Yann, Bengio, Yoshua, and Hinton, Geoffrey. Deep learning. *Nature*, 521(7553):436–444, 2015.

Levine, Sergey, Finn, Chelsea, Darrell, Trevor, and Abbeel, Pieter. End-to-end training of deep visuomotor policies. *Journal of Machine Learning (JMLR)*, 17:1–40, 2016.

Lipton, Zachary C., Gao, Jianfeng, Li, Lihong, Li, Xiujun, Ahmed, Faisal, and Deng, Li. Efficient exploration for dialog policy learning with deep BBQ network & replay buffer spiking. *CoRR*, abs/1608.05081, 2016.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Ng, A. Y. and Russell, S. J. Algorithms for inverse reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2000.

Piot, B., Geist, M., and Pietquin, O. Learning from demonstrations: Is it worth estimating a reward function? In *European Conference on Machine Learning (ECML)*, 2013.

Piot, B., Geist, M., and Pietquin, O. Boosted bellman residual minimization handling expert demonstrations. In *European Conference on Machine Learning (ECML)*, 2014a.

Piot, Bilal, Geist, Matthieu, and Pietquin, Olivier. Boosted and Reward-regularized Classification for Apprenticeship Learning. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2014b.

Ross, S., Gordon, G. J., and Bagnell, J. A. A reduction of imitation learning and structured prediction to no-regret online learning. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2011.

Schaal, Stefan. Learning from demonstration. In *Advances in Neural Information Processing Systems (NIPS)*, 1996.

Shani, Guy, Heckerman, David, and Brafman, Ronen I. An mdp-based recommender system. *Journal of Machine Learning Research*, 6:1265–1295, December 2005. ISSN 1532-4435.

Silver, David, Huang, Aja, Maddison, Chris J., Guez, Arthur, Sifre, Laurent, van den Driessche, George, Schrittwieser, Julian, Antonoglou, Ioannis, Panneershelvam, Veda, Lanctot, Marc, Dieleman, Sander, Grewe, Dominik, Nham, John, Kalchbrenner, Nal, Sutskever, Ilya, Lillicrap, Timothy, Leach, Madeleine, Kavukcuoglu, Koray, Graepel, Thore, and Hassabis, Demis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529:484–489, 2016.

Suay, Halit Bener, Brys, Tim, Taylor, Matthew E., and Chernova, Sonia. Learning from demonstration for shaping through inverse reinforcement learning. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2016.

Subramanian, K., Jr., C. L. Isbell, and Thomaz, A. Exploration from demonstration for interactive reinforcement learning. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2016.

Sutton, Richard S and Barto, Andrew G. *Introduction to reinforcement learning*. MIT Press, 1998.

Syed, U. and Schapire, R. E. A game-theoretic approach to apprenticeship learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2007.

Syed, U., Bowling, M., and Schapire, R. E. Apprenticeship learning using linear programming. In *International Conference on Machine Learning (ICML)*, 2008.

Taylor, M.E., Suay, H.B., and Chernova, S. Integrating reinforcement learning with human demonstrations of varying ability. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2011.

van Hasselt, Hado, Guez, Arthur, and Silver, David. Deep reinforcement learning with double Q-learning. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2016.

van Hasselt, Hado P., Guez, Arthur, Hessel, Matteo, Mnih, Volodymyr, and Silver, David. Learning values across many orders of magnitude. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

Watter, M., Springenberg, J. T., Boedecker, J., and Riedmiller, M. A. Embed to control: A locally linear latent dynamics model for control from raw images. In *Advances in Neural Information Processing (NIPS)*, 2015.

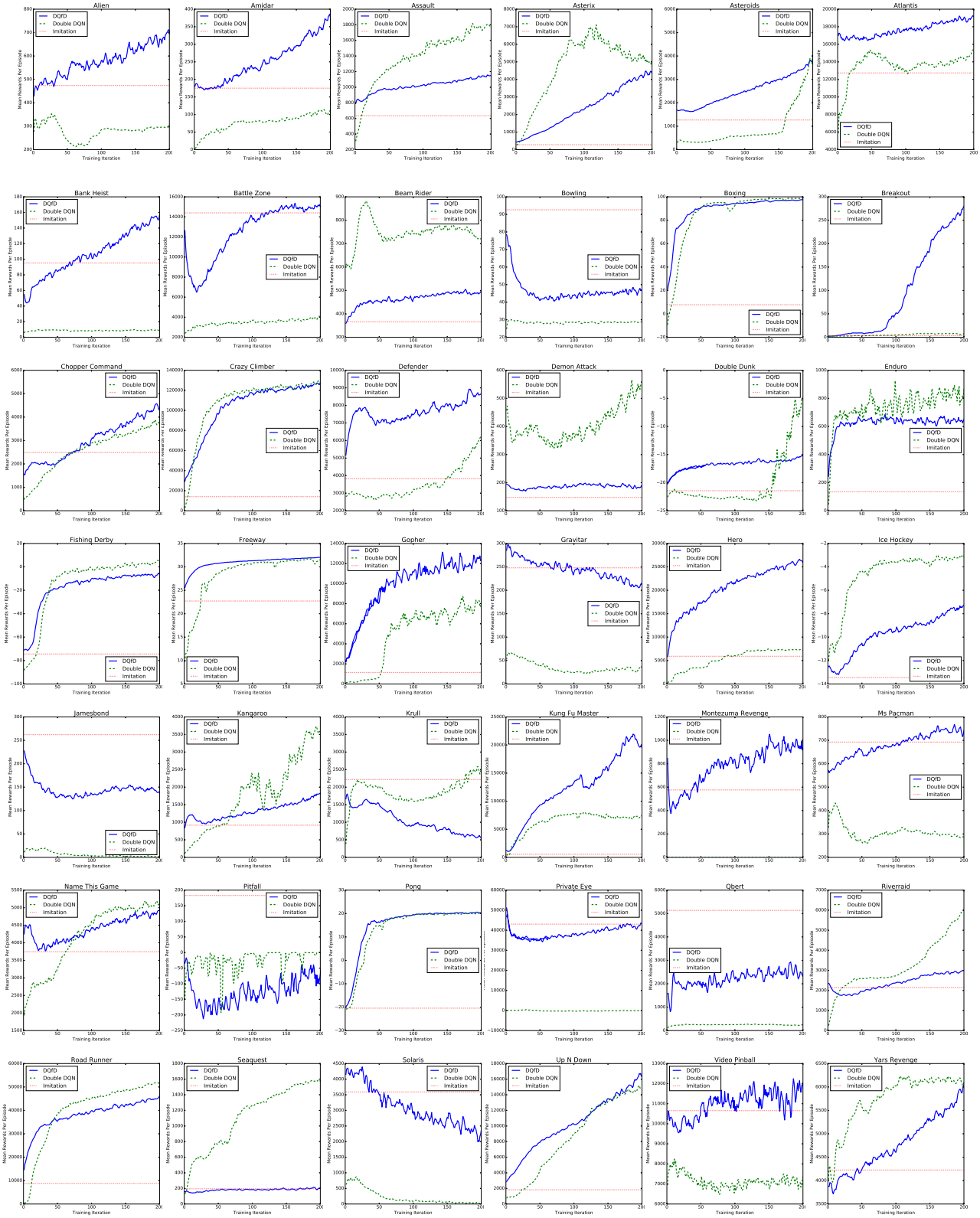

Figure 7. On-line rewards of the three algorithms on each of the 42 Atari games, averaged over 4 trials.