How to Make Reinforcement Learning Agents Learn Quicker?

Sai Prasath S
Georgia Institute of Technology
903840274
ss651@gatech.edu

Kritika Mittal
Georgia Institute of Technology
903566928
kmittal37@gatech.edu

Vitaly V. Marin
Georgia Institute of Technology
902267961
vmarin3@gatech.edu

***Abstract*—Reinforcement Learning (RL) agents are trained in an online setting where they interact with the environment for learning the optimal policy by exploring the state space. However, training an RL agent from scratch is usually time consuming, and does not make any use of expert's knowledge of the task. Utilizing expert demonstrations for bootstrapping an RL agent before the online learning phase has shown to achieve faster convergence and better overall performance in terms of cumulative rewards. However, how the expert demonstrations can be used during the online RL phase has not been studied. In this project, we analyze various strategies for using the expert demonstrations during online RL and derive rules/guidelines that are generalizable across different environments. We test these strategies on two different game environments namely: Cart Pole and Acrobot. From our analysis, we find that a 25% constant sampling of expert demonstrations while training provided the fastest convergence and overall performance.**

***Index Terms*—Reinforcement Learning, Bootstrapping, DQfD, DDQN**

I. Introduction

Reinforcement Learning agents learn by interacting with the environment in an online setting. At each state, the RL agent performs an action and receives a reward based on how good or bad the action was. Using the reward feedback, the parameters of the RL model are updated periodically to maximize the discounted sum of rewards acquired by the RL agent. The above steps are performed continuously until the RL agent converges, meaning, the parameter of the RL agent does not change by much, or the discounted sum of rewards acquired by the RL agent has plateaued [2]. However, training an RL agent from scratch for successfully completing a task is usually time consuming as the agent explores the whole state space to learn the optimal policy [7]. Moreover, the training process does not utilize any expert knowledge of the task.

In contrast, Learning from Demonstration (LfD) [1] techniques completely rely on expert knowledge of the task to learn a policy. In Behaviour Cloning (BC), a particular LfD technique, the policy is learnt by forcing the model to follow the trajectories (state-action pairs) of the expert demonstrations. As BC does not involve exploration, and only uses expert demonstrations to learn a policy, it is usually more sample efficient and quicker than RL algorithms. However, the quality of the policies learnt by BC is significantly limited by the quality of expert demonstrations, which are not always optimal [1]. Therefore, finding the right balance between using expert demonstrations to learn the policy and using online RL to explore the environment is crucial to ensure better overall performance, faster convergence, and sample efficiency.

Most RL models take a lot of time (thousands of episodes) before the performance of the models starts to improve. When access to a good simulator is available this is usually not a problem. However, for many real-world applications good simulators can be really difficult to build. Hence, the RL models might have to be deployed in real-world where the consequences of bad actions (during the initial stage) can be significant. While good simulators might not be readily available, most RL models have access to expert data for the task. Using these expert demonstrations to bootstrap an RL agent with BC before online RL has shown to boost the initial performance of the model and help avoid the initial phase where the performance of the RL agent is poor [7]. It has also shown to achieve better overall performance and faster convergence.

However, how the expert demonstrations must be used during the online RL phase has not been explored. In this project work, we analyze various strategies for using the expert demonstrations during the online RL phase such as linear annealing, exponential annealing, threshold based annealing, and constant sampling, and test their performance on two different game environments, namely: Cart Pole and Acrobot. We show that:

- A low percentage of expert demonstrations, with larger space in the buffer for exploration, should be present during training to guide the RL agent to converge faster.
- Suddenly dropping the sampling rate to 0% makes the model unstable and makes the agent forget, before it re-learns again.
- 25% constant sampling of the expert demonstrations during training is the best technique to help an RL agent converge faster.

II. Related Works

In 2018, Hester et. al. proposed the Deep Q Learning from Demonstrations (DQfD) where the Double Deep Q Network (DDQN) was pre trained on expert demonstrations using a large margin supervised loss which forces the network to mimic the expert's actions over the other actions [7][3]. During online RL, the model uses both the expert demonstrations and the generated trajectories for updating the model parameters. For computing the gradients for model update, the model uses both the large margin supervised loss and the Q-learning loss for the expert demonstration samples, but only uses the Q-learning loss for the the generated trajectories. This helps the model find a balance between imitating the expert demonstrations, and simultaneously improving the policy using RL exploration. In this project, we use the DQfD model for our experiments.

The model also uses only one replay buffer to store both the expert demonstrations and trajectories generated during online RL, but samples the expert demonstrations with higher priority during the model updates, and the priority remains constant throughout the whole process. Similarly, Pohlen et al. 2018 proposed the Ape-X DQfD model where multiple RL agents are run in parallel to generate trajectories and an independent learner uses the generated trajectories along with the expert demonstrations to learn an optimal policy [9]. The Ape-X DQfD model uses separate buffers to store the expert demonstrations and generated trajectories, but samples a constant 25% from the expert demonstrations buffer for each model update throughout the whole process. However, neither of these works provide any kind of empirical or theoretical justification for their choice. Hence, in this project we empirically evaluate various strategies for using expert demonstrations during the online RL phase and study how they affect the overall performance and the initial learning rate of the models.

Hosu et al. 2016 proposed the *human experience replay* technique where the RL agent stores the expert demonstrations in a separate replay buffer and obtains 50% of required samples from this buffer for each model update [5]. However, the model is not pre-trained on these expert demonstrations and no specific supervised loss is used during the gradient computations. The proposed technique was tested on Montezuma's Revenge and Private Eye, which are two of the most difficult games in Atari, but the performance was only slightly better than a random policy because of the sparsity of rewards in the games. They identified that although *human experience replay* provides better exploration compared to random agent, without bootstrapping the agent is not able to achieve better performance. The 50% sampling rate used in this research work is not explained and multiple sampling rates have not been evaluated in this work. Moreover, Lipton et al. 2017, analyzed how spiking the replay buffer (adding expert demonstrations to the replay buffer before online RL) affects the performance of their model [8]. They found that spiking the replay buffer helped models converge without which they otherwise failed. Also, they found that increasing the number of expert demonstrations in the replay buffer lead to the models learning quicker initially but at the cost of overall performance. However, these results are specific to a movie booking task. In our project, we test the various strategies across different environments, to derive generalizable rules/guidelines that can be extended to other environments.

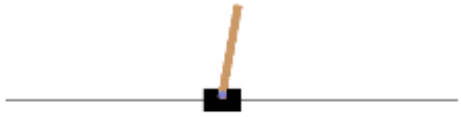

Fig. 1. Cart Pole Environment

III. Methods

A. Environment Setup:

In our work we used two classic control environments, Cart Pole and Acrobot from OpenAI gym [4].

Cart Pole environment: A pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. The pendulum is placed upright on the cart and the goal is to balance the pole by applying forces in the left and right direction on the cart.

Action space is an array with shape (1,) which can take values 0, 1 indicating the direction of the fixed force the cart is pushed with.

State space is an array with shape (4,) with different velocity values.

Reward: Since the goal is to keep the pole upright for as long as possible, a reward of +1 for every step taken, including the termination step, is allotted. The threshold for rewards is 500.

Termination: The episode ends if any one of the following occurs: (1) Pole Angle is greater than ±12°, (2) Cart Position is greater than ±2.4 (center of the cart reaches the edge of the display), (3) Episode length is greater than 500.

The Acrobot environment: The system consists of two links connected linearly to form a chain, with one end of the chain fixed. The joint between the two links is actuated. The goal is to apply torques on the actuated joint to swing the free end of the linear chain above a given height while starting from the initial state of hanging downwards.

Action Space is an array with shape (3,) which can take values 0, 1, 2 corresponding to applying torques -1, 0, and 1 to the actuated joint.

Fig. 2. Acrobot Environment

State Space is an array with shape (6,) that provides information about the two rotational joint angles as well as their angular velocities.
Rewards: The goal is to have the free end reach a designated target height in as few steps as possible, and as such all steps that do not reach the goal incur a reward of -1. Achieving the target height results in termination with a reward of 0. The reward threshold is -100.
Termination: The episode ends if one of the following occurs: (1) The free end reaches the target height, which is constructed as: -cos(theta1) - cos(theta2 + theta1) less than 1.0. (2) Episode length is greater than 500.

B. Model Architecture:

Deep Q Learning from Demonstrations: In our project, we use the DQfD architecture proposed in Hester et al. 2018 but with separate buffers for the expert demonstrations and generated trajectories [7]. The model uses the loss function as defined in 1.

$$J(Q) = J_{DDQN}(Q) + \lambda * J_S(Q) \tag{1}$$

The loss function is a combination of the DDQN loss and the supervised loss (for the expert demonstrations) with λ as the weighting parameter. The DDQN loss function in 2 is similar to the DQN loss function, except that we use Q_{target} to compute the values for the next state action pair (s', a') instead of Q, which helps in addressing the overestimation problem of DQNs [6]. The supervised loss function in 3 forces the model to mimic the expert demonstrator's actions. The function $l(a_E, a) = 0$ if $a_e = a$ and positive otherwise. Therefore, to minimize this loss the model is encouraged to take actions that are similar to the expert demonstrator.

$$J_{DDQN}(Q) = [Q(s,a) - \gamma * (r + \max_{a'} Q_{target}(s', a'))] \tag{2}$$

$$J_S(Q) = \max_{a \in A}[Q(s,a) + l(a_E, a)] - Q(s, a_E) \tag{3}$$

For the bootstrapping process, we only use the supervised loss function defined in 3. However, for the online RL phase we use both the DDQN loss and the supervised loss for the samples from the expert demonstrations buffer and only the DDQN loss for samples from the generated trajectories buffer. Algorithm 1 and figure 3 describes the whole bootstrapping and online RL learning process.

Algorithm 1: Deep Q Learning from Demonstration

Input: D_{expert}: Initialized with expert demonstrations, $D_{generated}$: Empty, θ : weights for the model (random), θ' : weights for the target network (random), τ : target model update frequency, k : number of pretraining steps, m : number of training episodes, $f(.)$: sampling rate function

***Bootstrapping Phase*:**
for *steps* $t \in \{1, 2, 3, .., k\}$ **do**
  Sample a mini-batch of size n *from* D_{expert}
  Calculate loss $J(Q)$
  Perform a gradient descent step on θ
  if t *mod* τ = 0 ***then*** $\theta' \longleftarrow \theta$ ***end if***
end

Online RL Phase:
for *steps* $t \in \{1, 2, 3, .., m\}$ **do**
  Sample action $a \sim \pi_{Q_\theta}$
  Perform action a *and observe* r *and* s'
  Store (s, a, s', r) *into* $D_{generated}$*, overwriting the oldest data store if exceeded capacity (FIFO)*
  Sample a mini-batch of size $n * f(t)$ *from* D_{expert}
  Sample a mini-batch of size $n - n * f(t)$ *from* $D_{generated}$
  Calculate loss $J(Q)$
  Perform a gradient descent step on θ
  if t *mod* τ = 0 ***then*** $\theta' \longleftarrow \theta$ ***end if***
  $s \longleftarrow s'$
end

Generating Expert Demonstrations: For bootstrapping the model we need access to expert demonstrations. However, playing the game multiple times to generate the demonstrations is time consuming. Hence, for our project we train a DDQN model to generate the expert demonstrations. We run the DDQN model for the games until convergence (no bootstrapping or expert demonstrations) and store the model weights. We use the converged model to generate trajectories and use those trajectories as the expert demonstrations for our DQfD model.

C. Annealing Techniques:

In our project, we analyze four different strategies for utilizing the expert demonstrations during the online RL phase. These strategies vary the sampling rate function f described in algorithm 1 which affects the number of samples drawn from D_{expert} for model updates. The four strategies implemented are as follows:

- **Linear Annealing:** In this technique we start the sample rate from a value $initial\_rate$ and reduce the value by $decay\_rate$ for each episode.

$$f(t) = max(0, initial\_rate - decay\_rate * t) \tag{4}$$

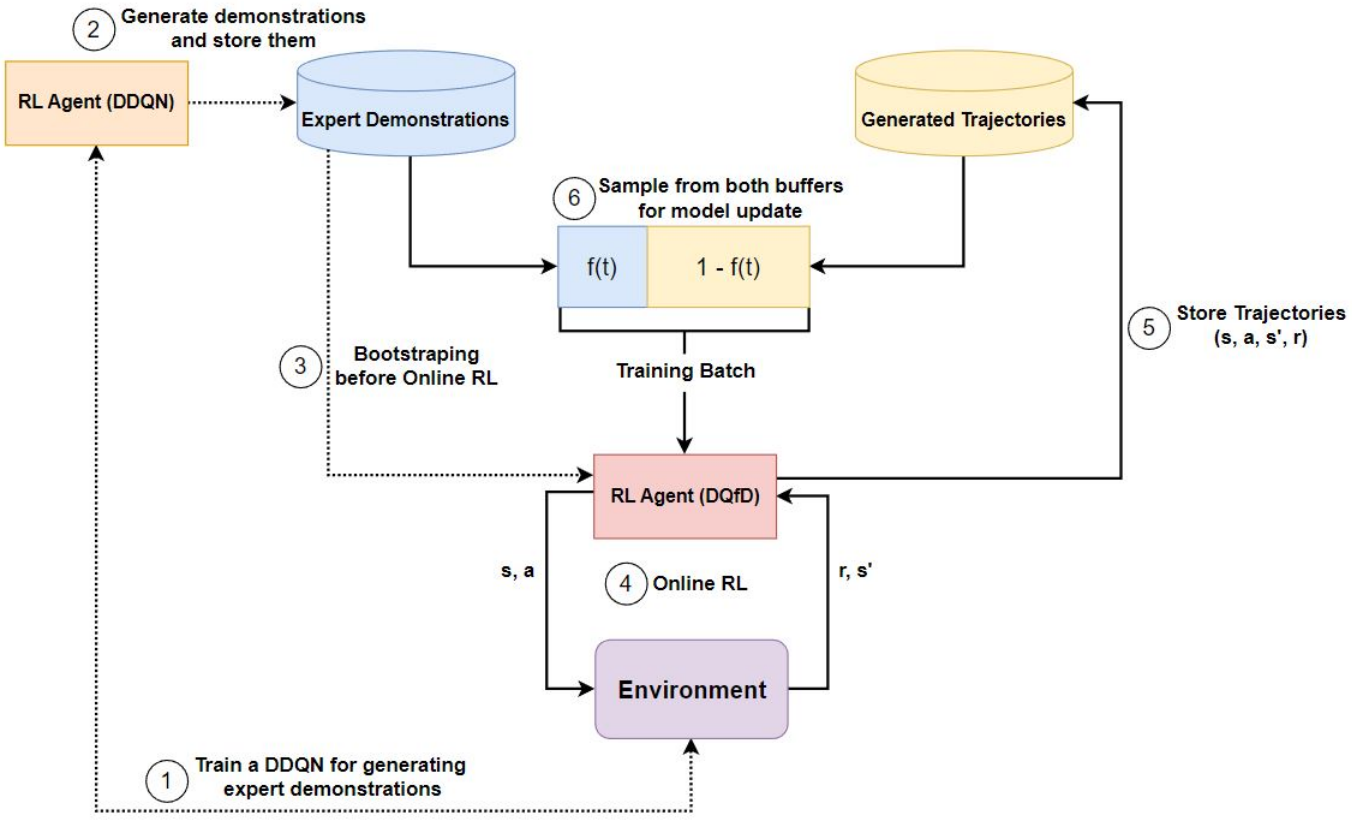

Fig. 3. DQfD Model Architecture

- **Exponential Annealing:** In this technique we start the sample rate from a value $initial\_rate$ and reduce the value exponentially at a $expo\_const$ rate.

$$f(t) = initial\_rate * e^{-expo\_const*t} \tag{5}$$

- **Threshold Based Annealing:** In this technique we use the sample rate from a value $initial\_rate$ if the average 10 episode reward is lesser than $anneal\_threshold$. Else we use a sample rate 0.

$$f(t) = \begin{cases} initial\_rate & \text{if 10-episode avg reward} \\ & < anneal\_threshold \\ 0 & \text{else} \end{cases} \tag{6}$$

- **Constant Sampling:** In this technique we maintain a constant sampling rate of $initial\_rate$ throughout the whole process. This is the technique used in [9] and [5], and therefore serves as our baseline.

$$f(t) = initial\_rate \tag{7}$$

IV. Experimental Results and Discussion

Experiment Setup: In this project, we implement and analyze the four annealing techniques discussed in section III C for different parameter values. We first execute these techniques for the Cart Pole game using the parameter values described in Table I. Then, by observing how the initial learning and overall performance of the model is affected by the annealing techniques we propose hypothesis that defines these behaviours. To validate our hypothesis, we test them on the Acrobot game. Apart from the parameters of the annealing techniques, there are other RL parameters such as learning rate, epsilon (exploration parameter), batch size, and gamma (discount rate) which needs to be fine tuned for each annealing technique. However, due to the lack of time and compute we tune these parameters for the DDQN network which is used for generating expert demonstrations and use those parameters for all experiments. We run three seeds for each paramter combination for both the games. For all the results we observe the initial learning rate of the technique and the overall performance.

A. Constant Sampling:

From figure 4 we can observe that when using a sampling rate of 100% for the expert demonstrations buffer, although there is slight improvement in the performance initially, the overall performance of the agent is not impacted and stays almost the same. For sampling rates of 50% and 75% we observe that the initial learning is slow, with the former technique improving rapidly around 60 episodes and the latter improving gradually throughout. With no samples from the expert demonstration (sampling rate of 0%), there is a slight

TABLE I
PARAMETER VALUES FOR CART POLE AND ACROBOT

	Initial Rate	Decay Rate	Expo Const	Anneal Threshold
Linear	0.25, 0.5	0.0025, 0.005, 0.01		
Exponential	0.25, 0.5		0.001, 0.01, 0.1	
Threshold Based	0.25, 0.5			C : 100, 200, 300 / A: -150, -200
Constant	0.0, 0.25, 0.5, 0.75, 1.0			

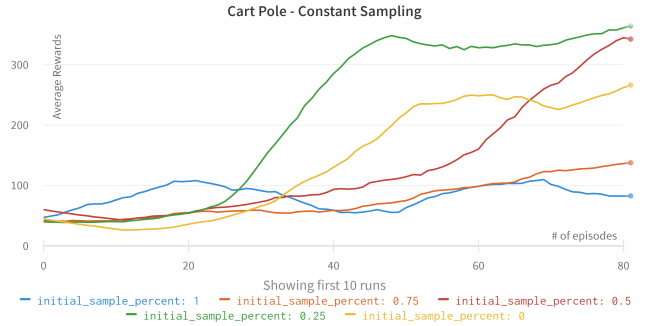

Fig. 4. Cart Pole - Constant Sampling

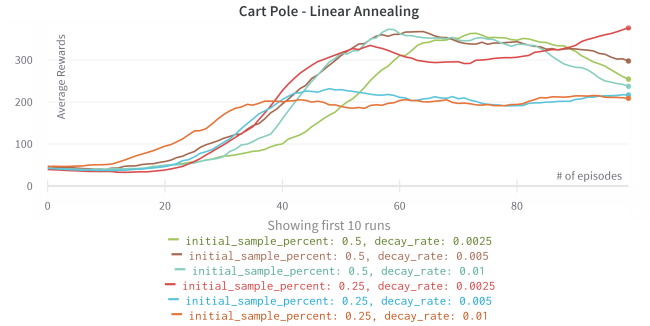

Fig. 5. Cart Pole - Linear Annealing

decrease in the initial performance before the learning starts. We observe the best overall performance with a sampling rate of 25% with rapid learning around 25 episodes and convergence around 50 episodes.

We believe that the technique with 100% sampling is similar to behaviour cloning and imposes a restriction on learning beyond the expert demonstrations. Similarly for 50% and 75% sampling rates, we observe a high reliance on the expert demonstrations and lack of exploration during the initial training which could have attributed to the slower learning. The initial drop in performance for no sampling could be because of forgetting. Because the agent no longer has access to the expert demonstrations on which bootstrapping was done, it tends to undo that learning before it starts learning again. 25% sampling seems optimum because it has sufficient expert demonstrations to guide the RL agent while having enough buffer space to allow the agent to explore and surpass the expert demonstrations. From the results we hypothesize that **a low percentage of expert demonstrations should be present during training to guide the RL agent while not inhibiting exploration.**

B. Linear Annealing:

From figure 5, we can observe that at the end of 40 episodes all the techniques which uses an initial sampling rate of 25% has better performance compared to the techniques that use an initial sampling rate of 50%. However, for the two techniques with an initial sampling rate of 25% and higher decay_rate = {0.005, 0.01} the performance plateaued around episode 50 and their overall performance is worse than the other linear annealing techniques. We observe that other techniques whose performance did not plateau either had a higher initial sampling rate of 50% or a slower decay_rate of 0.0025.

C. Exponential Annealing:

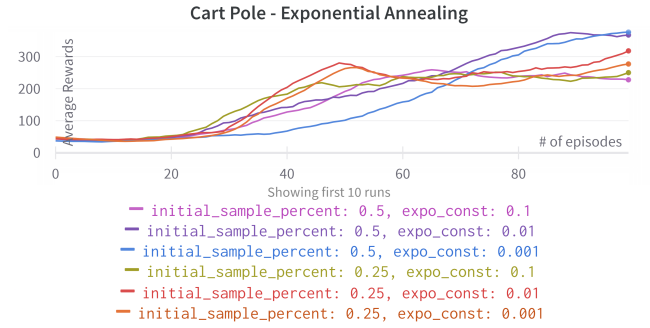

Fig. 6. Cart Pole - Exponential Annealing

From figure 6, we can observe that at the end of 40 episodes all the techniques with uses an initial sampling rate of 25% has better performance compared to the techniques that use an initial sampling rate of 50%. This fits our first hypothesis. However, all the techniques which start from an initial sampling rate of 25%, and the technique with 50% sampling rate and a high expo_const = 0.1 plateaued around episode 60 and performed worse than other exponential annealing technique. We also observe that the two techniques which achieved the highest performance had a 50% initial sampling rate and lower expo_const = {0.01, 0.001}.

Comparing the linear and exponential annealing techniques, we find that techniques that start from an higher initial sampling rate and has slower decay (low decay_rate or low expo_const) tends to achieve better overall performance as their sampling rate remains relatively higher throughout the

learning process. However, the high overall performance comes at the cost of initial learning rate, where most of these techniques lag behind other techniques for the first 40-60 episodes. Hence, we hypothesize that **using a higher initial sampling rate or a lower decay rate can help achieve better overall performance at the cost of slower initial learning rate**.

D. Threshold Based Annealing:

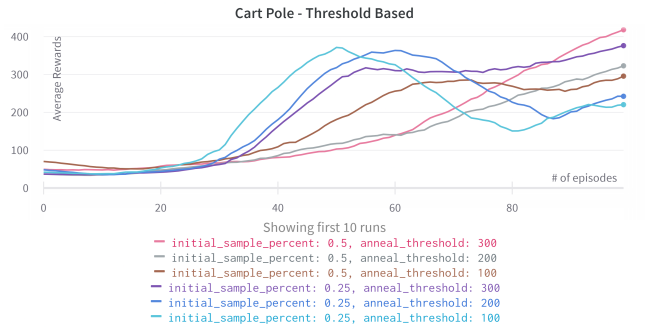

Fig. 7. Cart Pole - Threshold Based Annealing

From figure 7, we can observe that at the end of 40 episodes all the techniques which uses an initial sampling rate of 25% has better performance compared to the techniques that use an initial sampling rate of 50%. Two techniques with initial sampling rate of 25% and anneal threshold = {100, 200} achieve high performance around 40-60 episodes after their sampling rate is dropped to 0% after reaching the anneal threshold. However, soon after that their performance drops drastically. Similar behaviour can be observed with the technique that starts with an initial sampling rate of 50% and anneal threshold = 100, although the drop in performance is not so significant. All other models constantly improve their performance across all episodes without any drops, but we believe that these models also will suffer a drop in performance eventually. Similar behaviour was observed in figure 5 for initial sampling rate = 0% where the performance of the model initially drops before it starts to pick up again. Therefore, we hypothesis that **suddenly dropping the sampling rate to 0% makes the model unstable and makes the model forget, before it re-learns again**.

E. Analysis across different techniques

We select the best performance across the different annealing techniques to find the best overall technique. The techniques compared in Figure 8 are 25% constant sampling, linear annealing with 50% initial sampling and 0.5% decay, exponential annealing with 50% initial sampling and an exponential decay constant of 0.01 and threshold-based annealing with 50% initial sampling and -300 reward threshold. We observe that **the 25% constant sampling technique converges the fastest and has the best overall performance**

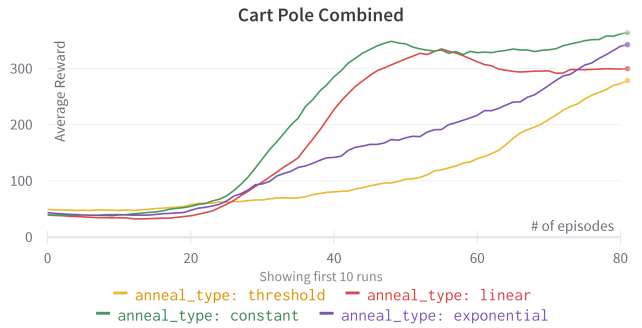

Fig. 8. Cart Pole - Different Sampling Techniques

while the threshold-based annealing technique performed the worst. This result is also coherent with our conclusions in the previous sections. The 25% constant sampling has a small expert demonstration buffer leaving room for exploration, it has a low decay, which is 0, and there is no abrupt decrease in the sampling rate which may cause forgetting.

F. Analysis on Acrobot

In this section, we test the following three proposed hypothesis by analysing the behaviours of various techniques on the Acrobot game.

- H1: A low percentage of expert demonstrations, with larger space in the buffer for exploration, should be present during training to guide the RL agent to converge faster.
- H2: Using higher initial sampling rate (with decay) or a lower decay rate can help achieve better overall performance at the cost of slower initial learning rate.
- H3: Suddenly dropping the sampling rate to 0% makes the model unstable and makes the model forget, before it re-learns again.
- H4: 25% constant sampling is the best technique.

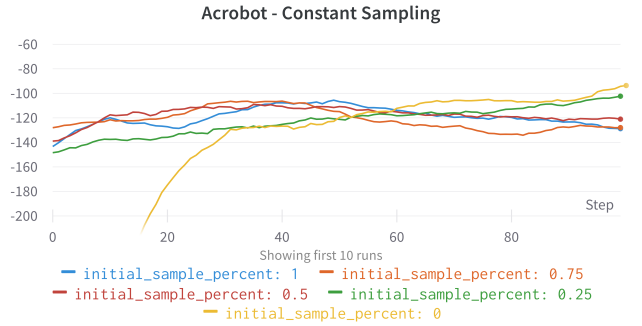

Fig. 9. Acrobot - Constant Sampling

H1: Figure 9 shows trends similar to those observed for Cart Pole. The technique with no sampling has a slower start than techniques with higher sampling rates. The policy converges faster for techniques with expert demonstrations

in the buffer. After 40 episodes, we observe that techniques with sampling rates 50%, 75% and 100% start dropping in performance. Although all policies with sampled expert demonstrations are starting very close to convergence, after 50 episodes we see 25% sampling outperforming the rest. We believe that the closer start to convergence, of the techniques with expert demonstrations in the buffer, can be attributed to the optimality of the expert demonstrations for Acrobot which is also evident from the overall performance of the 100% sampling technique. **Because of a high initial start and fast convergence of the policy with 25% sampling, we accept H1.**

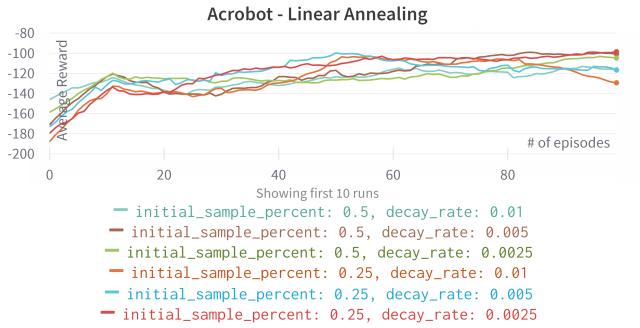

Fig. 10. Acrobot - Linear Annealing

H2: From Figure 10 we can observe that at the end of 40 episodes, 2 out of the 3 techniques that use a initial sampling rate of 25% perform the best. Although the other technique which uses a initial sampling rate of 25% with a decay_rate = 0.01 performs more exploration it lags behind two techniques with an initial sampling rate of 50% and decay_rate = {0.005, 0.0025} which perform the least exploration. Comparing the performance at the end of 100 episodes, we find that all the techniques that achieved the best performance for the Cart Pole linear annealing, achieved the best performance for Acrobot as well. The best performing techniques either had a high initial sampling rate of 50% or a low decay_rate = 0.0025. For achieving better overall performance the models need to keep sampling from the expert demonstrations, which is achieved by techniques starting from higher sampling rates or techniques where the decay rate is low. Also, we observe that better overall performance comes at the cost of initial learning, where the better performing techniques at the end of 100 episodes usually lag behind other techniques at the end of 40 episodes. Surprisingly, **guidance from the expert demonstrations are useful even after the model surpasses the performance of the expert demonstrations. The expert demonstrations help in guiding the model for achieving better performance.**

From figure 11, we can observe that at the end of 40 episodes, the best performing technique has a low initial sampling rate of 25% and high expo_const = 0.1, which means the model performs more exploration and hence is able to achieve high performance. However, the second and

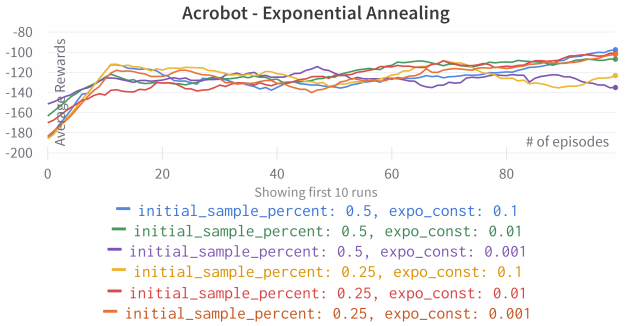

Fig. 11. Acrobot - Exponential Annealing

third best performing techniques have a high sampling rate of 50% with lower expo_const = {0.01, 0.001}. Despite relying heavily on expert demonstrations the techniques achieve faster initial learning. Although, some of this performance could be attributed to the quality of expert demonstrations which are significantly better than Cart Pole, no conclusion can be drawn. Similarly, at the end of 100 episodes, for technique with a sampling rate of 50% the ones with the higher expo_const perform better, whereas for techniques with a 25% sampling ratio the ones with the lower expo_const performs better. As per our hypothesis, the techniques that continuously rely on the expert demonstrations must achieve better overall performance. However, for the exponential annealing case no such conclusions could be drawn. **While the linear annealing techniques satisfy H2, for exponential annealing it is inconclusive. Hence, we need to perform more analysis to validate H2.**

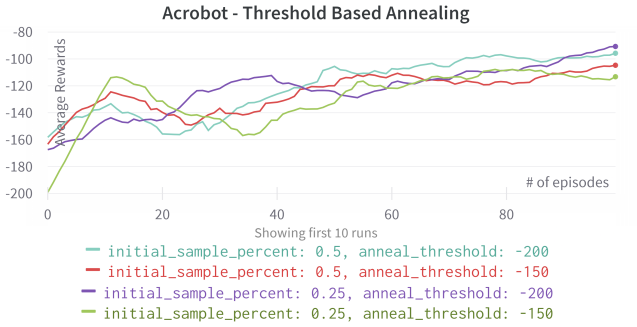

Fig. 12. Acrobot - Threshold Based Annealing

H3: From figure 12 we can observe that for both the techniques that use an anneal threshold of -150, the performance peaks around 15-20 episodes and significantly drops after that, and around 30-40 episode the performance starts to improve again as the model starts to re-learn. Similar behaviour can also be observed for the technique that uses a initial sampling rate of 25% and anneal threshold of -200, although the drop in performance is not significant. Also, comparing the initial performance of the 0% initial sampling

rates with other sampling rates in figure 9, we observe that the model performance drops significantly despite the bootstrapping, and the model starts learning after that. **As all the techniques analyzed for Acrobot fits H3, we accept H3.**

H4: The best runs of the different techniques are selected for comparison. 25% constant sampling, linear annealing with 50% initial sampling and 0.5% decay, exponential annealing with 50% initial sampling and an exponential decay constant of 0.01 and threshold-based annealing with 25% initial sampling and -200 reward threshold are shown in figure 13. **Like Cart Pole, 25% constant annealing has the best performance. Hence, we accept H4.**

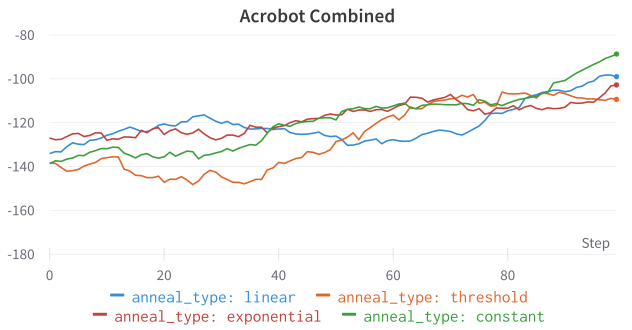

Fig. 13. Acrobot - Different Sampling Techniques

V. Limitations and Future Work

Due to time and resource constraints we were only able to validate our hypotheses for Cart Pole and Acrobot. Analysing these trends over other games is necessary for better generalizability. More testing is required to understand the trends across different games as well as across different hyperparameters which greatly impacted the rate of convergence of the DDQN. We only analysed games with a discrete action space and our results may not apply to games with a continuous action space. Further analysis can be done to understand the effect of the quality of expert demonstrations, used with the different annealing techniques, on the overall performance of the network. While the lower sampling rate may account for the suboptimality in the demonstrations, analysing the effect across the different techniques may provide some interesting results.

VI. Conclusion

We had initially expected that the RL agent will not need any expert demonstrations after sufficient learning and hence we tried out the different annealing strategies to gradually phase out the expert demonstrations during training. However, we found that constant sampling was the most stable. This is mainly because of the forgetting problem in reinforcement learning. It is important to have some amount of expert demonstrations always present in the replay buffer during training. This amount however needs to be small enough to allow exploration. From our experiments we found that the 25% constant sampling technique gave the best overall performance for the classic control environments, Cart Pole and Acrobot.

References

[1] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. "A reduction of imitation learning and structured prediction to no-regret online learning". In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2011, pp. 627–635.

[2] Volodymyr Mnih et al. "Playing atari with deep reinforcement learning". In: *arXiv preprint arXiv:1312.5602* (2013).

[3] Bilal Piot, Matthieu Geist, and Olivier Pietquin. "Boosted bellman residual minimization handling expert demonstrations". In: *Joint European Conference on machine learning and knowledge discovery in databases*. Springer. 2014, pp. 549–564.

[4] Greg Brockman et al. "Openai gym". In: *arXiv preprint arXiv:1606.01540* (2016).

[5] Ionel-Alexandru Hosu and Traian Rebedea. "Playing atari games with deep reinforcement learning and human checkpoint replay". In: *arXiv preprint arXiv:1607.05077* (2016).

[6] Hado Van Hasselt, Arthur Guez, and David Silver. "Deep reinforcement learning with double q-learning". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 30. 1. 2016.

[7] Todd Hester et al. "Deep q-learning from demonstrations". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1. 2018.

[8] Zachary Lipton et al. "Bbq-networks: Efficient exploration in deep reinforcement learning for task-oriented dialogue systems". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1. 2018.

[9] Tobias Pohlen et al. "Observe and look further: Achieving consistent performance on atari". In: *arXiv preprint arXiv:1805.11593* (2018).