Exploring demonstration pre-training with improved Deep Q-learning

Max Pettersson[1,2], Florian Westphal[2], Maria Riveiro[3]

***Abstract*— This study explores the effects of incorporating demonstrations as pre-training of an improved Deep Q-Network (DQN). Inspiration is taken from methods such as Deep Q-learning from Demonstrations (DQfD), but instead of retaining the demonstrations throughout the training, the performance and behavioral effects of the policy when using demonstrations solely as pre-training are studied. A comparative experiment is performed on two game environments, Gymnasium's Car Racing and Atari Space Invaders. While demonstration pre-training in Car Racing shows improved learning efficacy, as indicated by higher evaluation and training rewards, these improvements do not show in Space Invaders, where it instead under-performed. This divergence suggests that the nature of a game's reward structure influences the effectiveness of demonstration pre-training. Interestingly, despite less pronounced quantitative differences, qualitative observations suggested distinctive strategic behaviors, notably in target elimination patterns in Space Invaders. These retained behaviors seem to get forgotten during extended training. The results show that we need to investigate further how exploration functions affect the effectiveness of demonstration pre-training, how behaviors can be retained without explicitly making the agent mimic demonstrations, and how non-optimal demonstrations can be incorporated for more stable learning with demonstrations.**

I. INTRODUCTION

Reinforcement learning (RL) is a classification of both a problem domain and a set of solutions. It involves a problem domain where an agent interacts with an environment and, through rewards and punishments, searches for optimal strategies to achieve a goal [1]. An RL problem necessarily requires an environment that can be sensed by an agent, and allows for a goal that relates to the state of the environment. As such, the most common RL methods also require the agent to be able to perform actions to change states within the environment. There are four key aspects that an autonomous agent needs: an environment, sensation (or perception) of the environment, action to change states within the environment, and an agent goal [2]. Using a reward function defined within the environment, an agent will search for an optimal policy (set of actions) to maximize its accumulated rewards.

For RL problems, it is common to represent the environment as a Markov Decision Process (MDP), with a state space S, action space A, a transition function $P(s'|s,a)$, a reward function R, and a discount factor γ. The MDP is commonly represented as a tuple $M := (S, A, R, \gamma, P)$. More specifically, when the decision maker is in state $s \in S$, they choose an action $a \in A$ based on the current state. The MDP then probabilistically determines the next state $s' \in S$ and the reward $r \in R$ based on the current state s, action taken a, and the transition probabilities given by P. An RL agent will typically search through states and perform actions that maximize the expected reward. A collection of actions to perform for each state is typically referred to as a *policy* π, or more formally defined as any map $\pi : S \to A$ [3]. The policy that yields the highest expected cumulative reward is considered the *optimal policy* π^* [2].

An agent's exploration and learning function can be thought of in a cognitive decision-making framework and, more specifically, in terms of different learning strategies. Rendell et al. [4] review the idea of social learning strategies and explain that social learning is the strategy of learning from social information, which can be observations, interactions with other individuals, or its products. They contrast this to asocial learning and give trial and error as an example, which is analogous to the standard reinforcement models learning from scratch. They explain that copying strategies (social learning) from asocial learners (trial and error) is advantageous at a low-frequency rate; thus, they can avoid the cost of trialling the environment. Learning purely from trial and error is rarely a learning strategy employed by humans and animals [5]. Learning from copying strategies can be represented in the form of demonstrating desirable behaviors for the agent and is an idea that has proven successful for reinforcement learning [5], [3], [6], [7], and has spawned categories of algorithms like *imitation learning*, *learning from demonstrations*, *inverse reinforcement learning* etc.

In their seminal work, Mnih et al. [8] explain that RL agents have historically been limited to small problem domains where state representation and features could be handcrafted. Real-world problems usually have high-dimensional sensory inputs, and it is challenging to handcraft state representations that also generalize to new experiences from past experiences. Mnih et al. [8] created a Q-learning algorithm that uses a convolutional neural net for its state representation, DQN. They showed that by using a neural network to build abstract representations of raw image data, the DQN could generalize an environment representation good enough to learn and surpass human performance across 49 Atari games. This, among other early applications of neural nets, changed the viability of RL for more complex problems. Today, it is widespread to integrate neural networks in RL algorithms, and it has been shown to be able to solve problems that traditional RL has not [9].

Building upon the DQN algorithm, DQfD [6] incorporates

[1]M. Pettersson is a Ph.D student at Dept. of Computer & Information Science, Linköping University, Linköping Sweden, max.pettersson@liu.se

[2]Dept. of Computing, [3]Dept. Computer Science and Informatics, [2,3] both at Jönköping University, Jönköping, Sweden, florian.westphal@ju.se and maria.riveiro@ju.se

demonstrations into the algorithm. It leverages a small set of expert demonstrations to significantly improve the learning process, enabling the agent to start with an improved policy and continue improving through self-generated experience. This approach is particularly valuable in scenarios where agents must learn in real environments where the cost of exploration is high, and access to large amounts of simulation data is not feasible. DQfD demonstrates increased initial performance compared to agents learning from scratch, showcasing significant improvements on the first million steps in 41 out of 42 games tested. Furthermore, it achieves state-of-the-art performance in 11 games, underlining its efficacy in utilizing demonstrations for rapid learning.

The DQfD algorithm retains the demonstrations permanently throughout the whole training, and it guides the agent to mimic these demonstrations. However, exploring the effects of a simple pre-training with demonstrations, without retaining these demonstrations for the entire training duration, could shed light on whether initial exposure to demonstrations alone can influence the long-term learning trajectory and policy development of an agent. It may then be able to find novel behaviors outside the demonstrations that it would not otherwise when guided to mimic the demonstrations throughout the whole training.

This study examines the impact of demonstrations, solely as pre-training, on the behavior of an improved DQN agent that is similar to DQfD. The focus of this study is to test if the improvements to DQN with a pre-training with demonstrations can show improvements to the agent's training and, beyond the learning process, explore the behavioral changes that these demonstrations may induce. This comparison not only highlights the potential for expert demonstrations to guide the learning trajectory but also explores how augmentations to the Deep Q-learning affect agent behavior.

A. Related Works

RL agents can effectively learn from sparse or incomplete human demonstrations through various strategies. Brys *et al.* [10] and Nair *et al.* [7] both propose the use of reward shaping and demonstrations to speed up learning and overcome the exploration problem, respectively. Martínez *et al.* [11] introduce a model that requests teacher demonstrations only when they are expected to improve learning significantly and provides guidance to the teacher on which actions to demonstrate. Wang and Taylor [12] present the Dynamic Reuse of Prior (DRoP) algorithm, which combines offline knowledge with online performance analysis to achieve superior learning performance. These strategies collectively bridge the gap between demonstrated behaviors and exploring novel actions in RL.

Peng *et al.* [13] introduce what they call a goal-directed reinforcement learning framework for physics-based character animation. They demonstrate that natural character animations and behaviors can be imitated by an RL model through demonstrations of motions from motion capture data. The model could also learn complex control policies for novel scenarios while still accomplishing user-specified goals, e.g., imitating how to walk but also learning how to recover from external forces acting on the agent while walking (which was not present in the motion capture data). They contrast their agent's control policies to an agent that has been trained without motion capture data and show that training without motion capture data will cause the agent to solve the task in unnatural and unwanted ways. The example they show is an agent throwing a baseball as a human with their method, and without motion capture data the agent runs forward with the ball instead of throwing it.

II. Deep Q-Network and Deep Q-Learning from Demonstrations

A. Deep Q-Network

Algorithm 1 Deep Q-learning with Experience Replay. Adopted from [8].

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
for episode $= 1, M$ **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \arg\max_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set

$$y_j = \begin{cases} r_j & \text{if terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$$

    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ
    Every C steps reset $\theta^- = \theta$
  end for
end for

The DQN algorithm [8] consists of the Q-learning algorithm with a neural network for state space representation. Depending on the neural network architecture, it can also provide feature space representation and allows an agent to learn from the same features a human would, e.g. sending the game screen through initial convolutional layers of the neural network. In addition to the neural net, Mnih et al. [8] introduced three significant improvements to the algorithm to stabilize the training of the neural network.

Reward clipping was used to avoid Q-values getting too large and, as a result, avoid exploding gradients in the neural network. Secondly, they implemented something they call *"fixed target Q-network"*, which practically means that the algorithm has two neural nets with the same architecture, a target and a prediction network. The target network is a *"stale network"*, meaning it has fixed parameters that are not updated through gradient descent. Instead, the target network periodically updates its parameters by copying the prediction network's parameters. The prediction network provides the

state-by-state predictions, and its parameters are updated through gradient descent. The fixed target Q-network reduces oscillations in the training since the target network provides a moving target of a previous version of the network. The loss between the prediction from the prediction network and the fixed target Q-network is calculated for the backpropagation. Since the target Q-network updates its parameters with the parameters of the prediction network, the moving target will improve as the prediction network improves.

The final improvement is the implementation of an experience replay buffer (ERB). An experience is defined as $e_t = (s_t, a_t, r_t, s_{t+1})$, where t is the time-step. The ERB is a data set $D_t = (e_1, ..., e_t)$ created with experiences from multiple episodes, where an episode is an agent acting within the environment until a terminal state. In other words, the ERB is a memory where the agent stores a tuple of the transition between two states, what action it took to cause the transition, and what reward it got for the action. Batches of the ERB are randomly sampled to train the prediction network. The authors explain that this approach breaks the correlation between data points and thus reduces data inefficiency and variance in the updates. The ERB also allows for experiences to be used in multiple weight updates, which increases data efficiency.

B. DQfD

Algorithm 2 Deep Q-learning from Demonstrations. Adopted from [6].

Inputs: D^{replay}, initialized with demonstration data set, 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 D^{replay} with prioritization
    Calculate loss $J(\theta)$ using target network
    Perform a gradient descent step to update θ
    If $t \mod \tau = 0$ then $\theta^- \leftarrow \theta$ end if
end for
for steps $t \in \{1, 2, \ldots\}$ **do**
    Sample action from behavior policy $a \sim \pi_\theta$
    Play action a and observe (s', r)
    Store (s, a, r, s') into D^{replay}, overwriting oldest self-generated transition if over capacity
    Sample a mini-batch of n transitions from D^{replay} with prioritization
    Calculate loss $J(\theta)$ using target network
    Perform a gradient descent step to update θ
    If $t \mod \tau = 0$ then $\theta^- \leftarrow \theta$ end if
    $s \leftarrow s'$
end for

DQfD [6] uses a mix of demonstration data and data generated from the agent's own interactions with the environment. As seen in Algorithm 2, it initiates learning by filling the replay memory and pre-training on demonstration data to adopt an effective policy early on. It then continues to refine this policy with an enhanced DQN as it interacts with the environment.

In addition to the pre-training, DQfD uses Prioritized Experience Replay (PER) and a Dueling Network architecture (both are discussed in the next section) which are two improvements to the original DQN. The demonstration data is permanently stored in the replay memory. After the pre-training, it is used in conjunction with a supervised loss in order to ground the action values to imitate the demonstrations when the agent generates its own data. DQfD uses a weighted sum of three losses. Double Q-learning loss is a standard Temporal Difference loss but with an added n-Step improvement (as opposed to 1-step in regular DQN), meaning it considers n-step returns for a longer horizon of the reward estimate. Supervised Large Margin Classification Loss, which is calculated from the demonstration data to induce mimicking the demonstrations. L2 Regularization loss, to prevent overfitting to the demonstration data.

III. DQN Improvements

Multiple studies have suggested improvements to the DQN algorithm; here, we summarize the most relevant ones.

A. Double DQN

Double Q-learning [14], which is an improvement to the classical Q-learning algorithm, also proved useful for DQN [15]. Regular DQN has a tendency to overestimate Q-values for actions in certain situations because the max operator uses the same action value for selecting and evaluating actions. Double DQN reduces the overestimation of Q-values by separating the selection and evaluation of actions, practically this is done when calculating the targets for the backpropagation. For regular DQN, the target is calculated by

$$Y_t^{DQN} \equiv R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \theta_t^-)$$

In contrast, the double DQN target is calculated by

$$Y_t^{DoubleQ} \equiv R_{t+1} + \gamma Q(S_{t+1}, \arg\max_a Q(S_{t+1}, a; \theta_t); \theta_t^-)$$

The authors show that this decoupling of selection and evaluation of action values provides better, more stable training and better policies for larger-scale problems.

B. Prioritized experience replay

PER [16] is an enhancement of ERB based on improving the efficiency of sampling from the memory by adjusting the priority with which experiences are replayed. Traditional ERB replays random samples from the experience memory, which may not always be the most efficient method for learning. The main improvement in PER is to replay important transitions more frequently, based on the principle that some experiences are more important than others for learning. To

quantify the importance, the temporal difference (TD) error is used as a proxy, where transitions with high TD error are considered more significant and are thus replayed more often. A high TD error transition means that the model has a high error in its prediction of an action value, analogous to making the transition more surprising and more valuable to learn from.

This method of prioritization can introduce bias and lead to overfitting if used greedily. This is due to initial high TD error transitions getting replayed more frequently, which may only be a small subset of the memory. This can be mitigated through stochastic prioritization, which is a sampling method that interpolates between greedy TD prioritization and uniform random sampling and is given by

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

Where $p_i > 0$ is the priority of transition i. The exponent α determines the ratio of prioritization versus random sampling, where $\alpha = 0$ means only random sampling.

An additional bias occurs due to changing the distribution of sampling. One of the main ideas of DQN is to remove correlation from observations, which was achieved through uniform random sampling. Even with stochastic prioritization, there is still a bias of correlation with observations. To correct this bias, importance-sampling weights are used, given by

$$w_i = (\frac{1}{N} \cdot \frac{1}{P(i)})^\beta$$

Where $\beta = 1$ corresponds to the case of fully compensating for non-uniform probabilities.

In practice, PER involves:

1) **Storing Transitions:** As experiences are collected, they are stored in a replay buffer with their corresponding TD errors, which serve as their priorities.
2) **Sampling Transitions:** When selecting experiences for replay, transitions are sampled based on their priority or randomly, determined by stochastic prioritization.
3) **Updating Priorities:** After learning from a replayed transition, its priority is updated based on the new TD error, ensuring that the replay buffer reflects the current learning state of the agent.
4) **Correcting Bias:** To account for the non-uniform sampling, importance-sampling weights are applied to the learning updates to correct for the introduced bias.

C. Dueling Network

The Dueling Network architecture [17] introduces a neural network structure that separately estimates the state value function and the advantages for each action. It is designed to improve the learning of state value functions in environments where the state value does not significantly vary across actions. The Dueling Network divides the network into two streams that converge through an aggregating layer. One stream is responsible for estimating the state value function, providing a scalar value that represents the value of being in a particular state. The other stream estimates the advantage function for each action, indicating the relative importance of each action from that state. The final Q values, representing the value of taking an action in a given state, are obtained by combining these two streams.

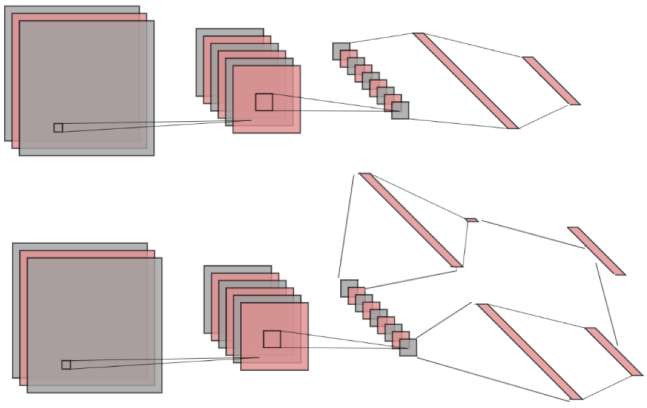


Fig. 1: Regular DQN (**top**) and dueling network (**bottom**). The dueling network splits into two streams where the output of the value stream is a single neuron, and the output of the advantage stream corresponds to the number of actions. For the final output, the two streams are combined into an output corresponding to the amount of actions.

D. Noisy Network

Noisy networks [18] introduces a novel method for improving exploration in DQN by integrating parametric noise directly into the weights of neural networks. This approach makes the agent perform exploration by inducing stochasticity in the agent's policy, where the parameters of the noise are optimized alongside the network's weights using gradient descent. Unlike traditional exploration techniques that rely on external noise sources or perturbations, Noisy networks achieve a state-dependent exploration strategy by affecting the network's internal parameters, leading to potentially complex changes in policy across different states.

Noisy networks are implemented by adding noise to both the weights and biases of the network, where the noise parameters are learned. This also allows for an automatic adjustment of the exploration intensity, removing the need for manually tuning exploration hyperparameters. The authors show that they achieve significant improvements across Atari games compared to DQN with and without the Dueling network.

IV. Method

For this paper, a comparative experiment is conducted in which two games are set up for the agent to learn from. Two models are trained on each game, one with demonstration pre-training and one without. For the demonstration model, the algorithm initiates with a pre-training phase utilizing demonstrations from a human playing the game, in order to

provide the agent with initial behavioral guidance. This pre-training ensures that the agent starts with a competent policy, reducing the initial exploration time required to achieve proficient performance.

A. Algorithm

Algorithm 3 Improved DQN with demonstration pre-training

Initialize PER D^{replay} to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
τ: frequency at which to update target net
Sequence $s = \{x\}$ and preprocessed sequence $\phi = \phi(s)$
k: number of episode demonstrations
for steps $t \in \{1, 2, \ldots, k\}$ **do**
    select $a_t =$ *human action*
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D^{replay}
    If step is terminal step then
    Sample a mini-batch of n transitions from D^{replay} with prioritization
    Perform a gradient descent step to update θ, end if
end for
$\theta^- \leftarrow \theta$
for episode $= 1, M$ **do**
    for steps $t \in \{1, 2, \ldots, m\}$ **do**
        select $a_t = \arg\max_a Q(\phi(s_t), a; \theta)$
        Execute action a_t in emulator and observe reward r_t and image x_{t+1}
        Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D^{replay}
        Sample a mini-batch of n transitions from D^{replay} with prioritization
        Set $y_j = R_{t+1} + \gamma \hat{Q}(S_{t+1}, \arg\max_a Q(S_{t+1}, a; \theta_t); \theta_t^-)$
        Perform a gradient descent step to update ϕ
        If $t \mod \tau = 0$ then $\theta^- \leftarrow \theta$ end if
    end for
end for

The algorithm used for this paper uses DQN as a foundation (Algorithm 1). It integrates the pre-training step of DQfD (steps 1-7 in Algorithm 2) as well as three of its improvements: Double DQN to mitigate overestimation bias by decoupling action selection and evaluation, PER to emphasize learning from transitions with higher expected learning utility, Dueling Network Architecture to refine the estimation of action values by distinguishing between state values and action advantages. A fourth improvement, Noisy networks is implemented to enhance exploration through the injection of parametric noise into the network weights. The inclusion of a Noisy network means that steps 7 and 8 in Algorithm 1 are removed. The PER memory is initialized with data from a human giving demonstrations in real-time. After the pre-training, the agent will start interacting with the game and fill the PER memory with its own transitions. The PER memory is a circular buffer, when the memory gets full, the demonstrations will be replaced with new transitions. The Neural Network structure for the Dueling Networks consists of three convolutional layers and two noisy layers, with rectified linear units used between all layers.

V. Experiment Setup

The experiments used two games, Car Racing and Atari Space Invaders, from the Python library Gymnasium [19].

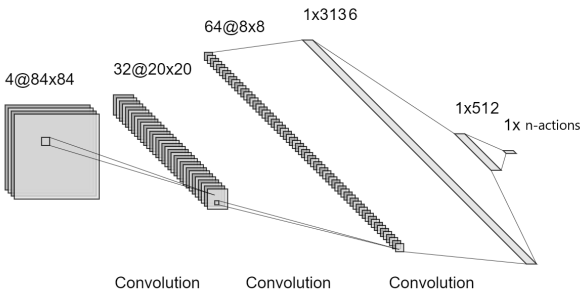
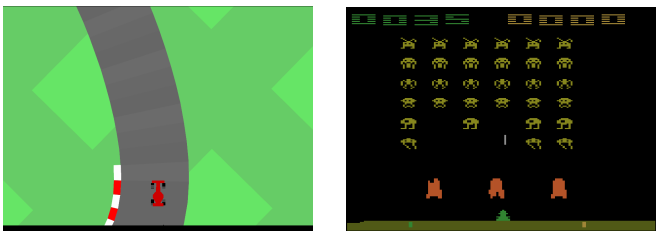


Fig. 2: Neural network structure of the algorithm. For the Dueling Network, two of these are defined, with the network for value approximation having an output of 1×1.

The Car Racing environment provides a dense reward function that gives positive rewards for almost every action if the optimal policy is followed. The Atari Space Invaders environment provides a less dense reward function, only giving rewards when the agent or player scores points in the game, which only happens when an invader is shot. This means that any change in positions, shots that do not hit an invader, or the agent losing a life does not provide any feedback.

For both games, the same four pre-processing steps are done on each game frame, similar to [8].

- **Frame skipping:** The algorithm applies an action to the game for four frames, but only every fourth frame is processed and stored in the PER, essentially skipping three frames. This is done to reduce computational time.
- **Grayscale conversion:** The RGB frame is converted into a grayscale one-channel image.
- **Downsampling:** The grayscale frame is converted to 84×84 pixels.
- **Frame stacking:** Four consecutive downsampled grayscale frames are stacked together. Practically, they are stacked as channels, meaning that for the convolutional layers, the final processed frame has a shape of $84 \times 84 \times 4$, where the four frames can be considered channels for the convolutions.



(a) The Gymnasium Car Racing game

(b) The Atari Space Invaders game

Fig. 3: The two games used for the experiment.

A. Car Racing

The Gymnasium race car environment is a game where a player or agent controls a car to navigate a track as quickly as possible. The environment presents a 2D top-down view of a race track (see Figure 3a), where the agent's goal is to complete the track. The track is randomized for each episode reset. For this game, the models were trained for 1 000 episodes using the hyperparameters stated in Table I.

- **Observation space:** An observation is a 96×96 pixel image (RGB) representing the agent's view of the environment. This view includes the car, the track, and the surrounding area.
- **Action space:** The action space contains five discrete actions, do nothing (NOOP), turn left (LEFT), turn right (RIGHT), and accelerate (GAS).
- **Episode Termination:** An episode ends when the car goes off the track or after 250 time steps.
- **Reward function:** The race track contains tiles that provide rewards for the agent when it crosses a tile. A time step penalty is also present in order to encourage faster completion. An accumulated episode reward of around 850–950 is considered a successful episode.

TABLE I: Hyper-parameters of both games

Hyper-parameter	Car Racing	Space Invaders
Learning rate α	$2.5 \cdot 10^{-5}$	$1 \cdot 10^{-3}$
Reward discount γ	0.9	0.9
Target network update frequency τ	5 000	10
PER alpha	0.2	0.5
PER beta	0.6	0.4
PER sample batch size	256	128
PER memory size	$1.0 \cdot 10^4$	$1.0 \cdot 10^5$

B. Atari Space Invaders

The Gymnasium Space Invaders environment is part of the Atari environment, which is a simulation of various Atari 2600 games (see Figure 3b). In this game, the player or agent controls a cannon at the bottom of the screen, aiming to shoot down waves of alien invaders moving horizontally across the screen while avoiding their attacks. The game stays the same for every episode, making it deterministic compared to Car Racing. For this game, the models were trained once for 10 000 episodes, and once for 20 000 episodes using the hyperparameters stated in Table I. This was done in order to investigate changes in behavior based on training time.

- **Observation space:** An observation is a 210×160 pixel image (RGB) representing the game screen, including the player's cannon, the invaders, the projectiles, the score, and the lives left.
- **Action space:** The action space contains six discrete actions, do nothing (NOOP), shoot (FIRE), move right (RIGHT), move left (LEFT), move right and fire (RIGHTFIRE), and move left and fire (LEFTFIRE).
- **Episode Termination:** An episode ends if the player or agent loses all lives.
- **Reward function:** The agent receives the game score as a reward. This means that it only gets rewards when an invader is shot. Since the termination of an episode only ends when all lives are lost, theoretically, the maximum reward is infinite. However, clearing the screen (shooting every invader) will net the player or agent a score of 630. For this study, getting a minimum of 630 is considered a successful episode.

VI. Results

As mentioned in the experiment setup, one comparison experiment was done on Car Racing for 1 000 episodes, and two comparison experiments were done for Space Invaders, one for 10 000 episodes and one for 20 000 episodes. Going forward, the Space Invaders models will be referred to as *10k* and *20k*, respectively. In total, 6 models were trained. During the training, rewards and actions were logged at each time step and then aggregated to their corresponding episode. Actions were also recorded for the demonstrations. Two metrics are used for learning performance analysis: the total reward per episode during training and an evaluation reward when letting the model play the game after being trained. For the Car Racing game, 50 evaluation episodes were run on each model and the mean is presented. For Space Invaders, only one evaluation episode was run, due to the deterministic nature of the game, every episode plays out the same during evaluation. For the behavioral analysis, the action distribution comparisons between the models and the demonstration will be used as a metric.

A. Learning performance

TABLE II: Model episode reward during evaluation

Game	Model	Evaluation reward
Car Racing	With pre-training	805
Car Racing	Without pre-training	718
Space invaders	With pre-training 10k	345
Space invaders	Without pre-training 10k	670
Space invaders	With pre-training 20k	495
Space invaders	Without pre-training 20k	545

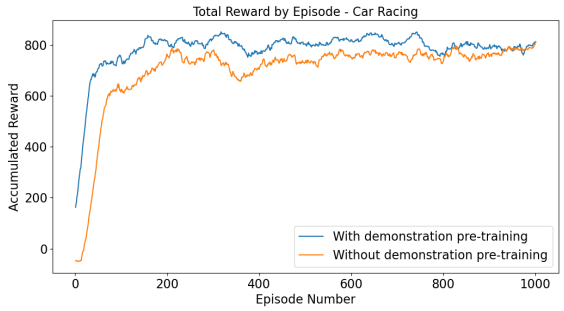


Fig. 4: Episode rewards during training for Car Racing.

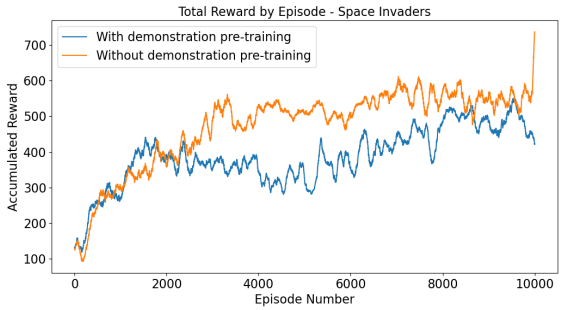


Fig. 5: Episode rewards during training for Space Invaders 10k.

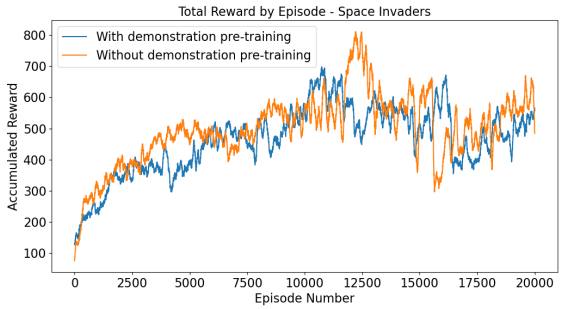


Fig. 6: Episode rewards during training for Space Invaders 20k.

B. Action distributions

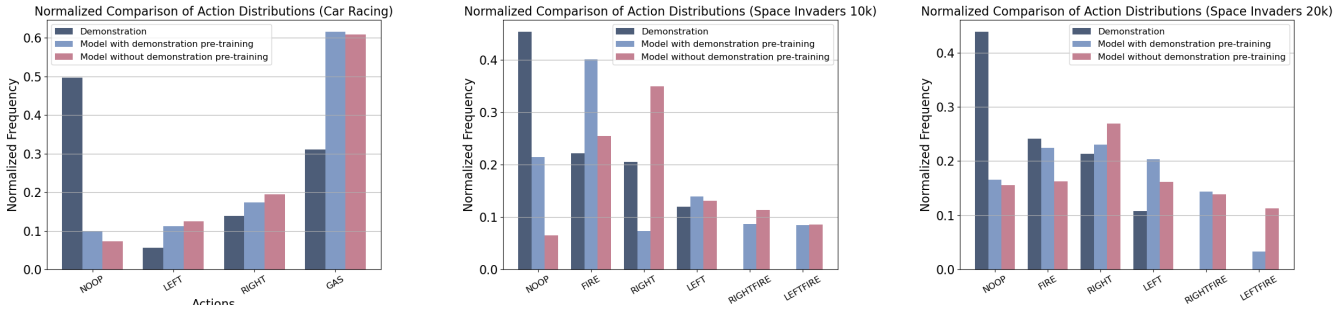
Figure 7 shows a normalized distribution of actions, with demonstration actions recorded from pre-training and model actions recorded during evaluation. Figure 8 shows kernel density estimations of actions during the Space Invaders evaluation episode, where the X-axis shows the time-step in the episode. Due to the stochastic nature of the randomized Car Racing game, a kernel density estimation is not provided. Figure 8a and 8d show a cutoff in the plot due to the demonstration not using the *RIGHTFIRE* and *LEFTFIRE* actions.

VII. Discussion

The overall results met initial expectations in some aspects but did not in others. Specifically, the outcomes for the Car Racing game align to some extent with the preliminary hypothesis that demonstration pre-training has a positive impact on learning performance as seen in Figure 4. Demonstration pre-training allowed the model to converge faster, and to a higher evaluation score after training, as shown in Table II. The same cannot be said for Space Invaders, which showed an overall worse training score, see Figure 5 and 6, and showed a clearly worse evaluation score, see Table II. This discrepancy might stem from the differences in reward structures across the two games; the Car Racing game frequently rewards actions, possibly amplifying the effectiveness of demonstrations. In addition, it was easier to perform a high scoring episode for the demonstration with Car Racing compared to Space Invaders. Contrary to expectations and differing from findings by Hester et al. [6], the introduction of demonstrations did not markedly improve performance compared to the non-demonstration approach and, in fact, worsened it. Although, in the DQfD approach, the demonstrations are kept throughout the training, and it is possible that better demonstrations were given, due to the authors stating they were using expert demonstrations. The demonstrations performed in this study were done by playing the games after some practice, the goal was not to try and achieve a proficient score, but to achieve a successful episode. In addition, the incorporation of a noisy network, which introduces stochasticity into the model's weights, could diminish the utility of demonstrations by perturbing the guidance they provide; this will be studied further in the future.

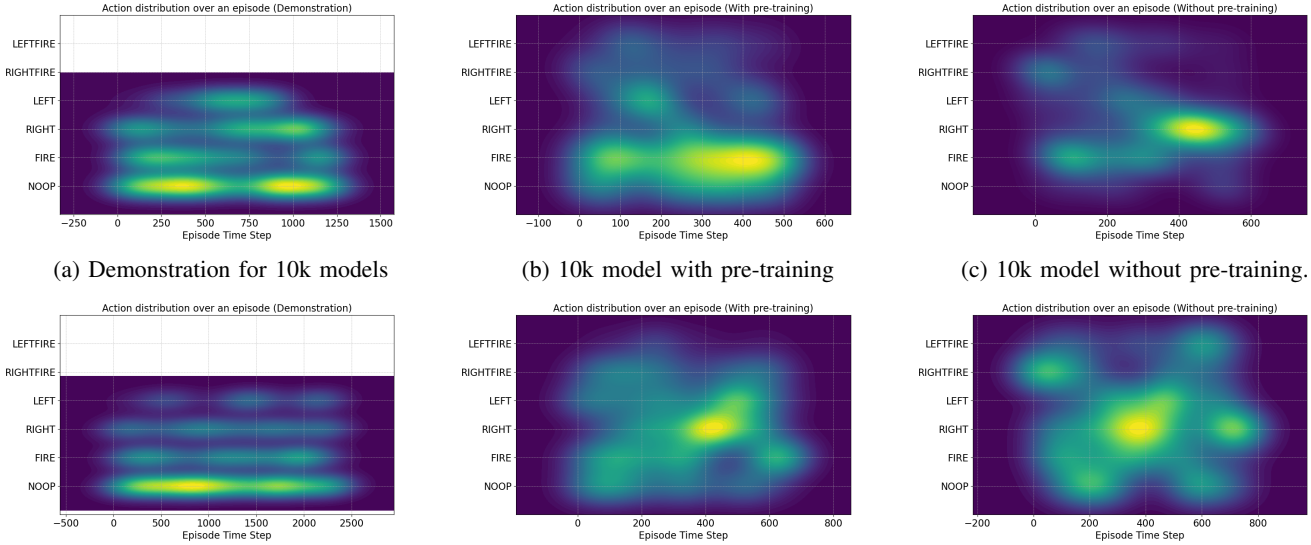
When it comes to the behavior of the agent, there are some interesting potential findings. Figure 7b shows that the 10k model without pre-training had a strong tendency to move right in the game. It also tended to fire less. The pre-training 10k model shows a tendency towards the demonstration when it comes to no action (NOOP) compared to no pre-training. The pre-training 10k model's action distribution seems to differ from the demonstration, but looking at the action density during an episode in Figure 8a and 8b, the action densities follow the demonstrations closer than the model without pre-training temporally, see Figure 8c. The FIRE and NOOP densities for the pre-trained model have a tendency towards following the demonstration densities, compared to the model without pre-trainng that has a high tendency to FIRE and RIGHTFIRE in the beginning and then go right without firing towards the end of the episode. For the 20k models, both the pre-trained and non pre-trained models seem to converge to similar action distributions (see Figure 7c) and densities (see bottom row of Figure 8). This could be due to the agent gradually forgetting the demonstration policies with extended training. Therefore, it is likely that any policies the agent learns from the pre-training will be more prevalent early in training. This, in conjunction with the reward mechanism of Space Invaders, may also explain why the pre-trained models generally perform worse than the non pre-trained model in the beginning, if the demonstrations are not optimal.

After qualitative analysis of watching the agent play the game, the models trained with demonstrations tended to eliminate alien spacecraft column by column in the early time-steps, following the strategy used in some of the demonstrations. Conversely, models trained without demonstrations preferred a row-by-row approach and ended up with a wider field of enemies towards the end. In addition, the Car Racing models seemed to show differences in behavior after a lap was completed, where the pre-trained models tended to go off the track after finishing a lap, while the model without pre-training tended to not. It may be the case that this behavior was retained due to the model stops receiving rewards after a lap is finished and thus does not receive feedback to over-write this behavior with a more optimal one. This is



(a) Car Racing. (b) Space Invaders 10k. (c) Space Invaders 10k.

Fig. 7: Histograms of action distributions. Figure 7a shows distributions for the Car Racing game, where similar policies between the models with and without pre-training can be interpreted. Figure 7b illustrates differences in action distributions between both models and demonstration, which indicates differences in policy. Although the distributions between the pre-trained model and demonstrations diverge, they indicate a closer similarity in overall behaviors compared to the model without pre-training and demonstration. Figure 7c presents distributions for models with longer training, and the two models show a larger similarity compared to the 10k models, suggesting that extended training pushes the two models to similar policies.



(a) Demonstration for 10k models (b) 10k model with pre-training (c) 10k model without pre-training.

(d) Demonstration for 20k models (e) 20k model with pre-training (f) 20k model without pre-training.

Fig. 8: Action densities for actions (y-axis) over an episode time-steps (x-axis). The figures on the top row show the densities of the 10k models, with the pre-trained model showing more similarities to the demonstration in FIRE and NOOP compared to the model without pre-training. The model without pre-training also indicates a preference for RIGHT action. The figures on the bottom show the action densities for the 20k models. These models' densities present closer similarity and a more uniform density distribution, most likely due to the longer training.

something that happened in some demonstrations for the Car Racing game, where the car was driven off the track after a lap. It should be noted that these are subjective interpretations of visually inspecting the agent playing, it may also be caused by randomness in training. These observations are mentioned as curiosities and should be seen as grounds for further investigation.

VIII. CONCLUSIONS

To conclude, demonstration pre-training alone may show improvements in learning and performance for an agent, but most likely, it depends on the quality of the demonstration and the reward mechanisms of the problem. Most likely, non-optimal demonstrations and less dense or delayed rewards may not provide the guidance for an initial policy to help the learning, and may, in fact, be a detriment to learning. Either incorporation of demonstration with reward function needs to be explicitly planned (as seen with DQfD [6]), or further investigation needs to be done on how to incorporate human-like non-optimal demonstration into the learning process in order to make demonstration pre-training more stable.

Demonstration pre-training has shown indications that the model retains behaviors of the demonstrations but has shown to lose these behaviors during extended training. It may be the case that these behaviors can be retained through other means than explicitly mimicking the demonstrations as long as the reward function allows it; this will be an interesting path for future investigations.

ACKNOWLEDGMENT

The authors acknowledge the Knowledge Foundation, Jönköping University, and the industrial partners for financially supporting the research and education environment on Knowledge Intensive Product Realization SPARK at Jönköping University, Sweden. Project: AFAIR with agreement number 20200223.

REFERENCES

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. The MIT Press, second ed., 2018.

[2] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3 ed., 2010.

[3] A. Y. Ng and S. J. Russell, "Algorithms for inverse reinforcement learning.," in *ICML* (P. Langley, ed.), pp. 663–670, Morgan Kaufmann, 2000.

[4] L. Rendell, L. Fogarty, W. J. Hoppitt, T. J. Morgan, M. M. Webster, and K. N. Laland, "Cognitive culture: theoretical and empirical insights into social learning strategies," *Trends in Cognitive Sciences*, vol. 15, pp. 68–76, 2 2011.

[5] S. Schaal, "Learning from demonstration," in *Advances in Neural Information Processing Systems* (M. Mozer, M. Jordan, and T. Petsche, eds.), vol. 9, MIT Press, 1996.

[6] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, I. Osband, G. Dulac-Arnold, J. Agapiou, J. Z. Leibo, and A. Gruslys, "Deep q-learning from demonstrations," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'18/IAAI'18/EAAI'18, AAAI Press, 2018.

[7] A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Overcoming exploration in reinforcement learning with demonstrations," *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6292–6299, 2017.

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

[9] H. nan Wang, N. Liu, Y. yun Zhang, D. wei Feng, F. Huang, D. sheng Li, and Y. ming Zhang, "Deep reinforcement learning: a survey," 12 2020.

[10] T. Brys, A. Harutyunyan, H. B. Suay, S. Chernova, M. E. Taylor, and A. Nowé, "Reinforcement learning from demonstration through shaping," in *International Joint Conference on Artificial Intelligence*, 2015.

[11] D. M. Martínez, G. Alenyà, and C. Torras, "Relational reinforcement learning with guided demonstrations," *Artif. Intell.*, vol. 247, pp. 295–312, 2017.

[12] Z. Wang and M. E. Taylor, "Interactive reinforcement learning with dynamic reuse of prior knowledge from human and agent demonstrations," in *International Joint Conference on Artificial Intelligence*, 2019.

[13] X. B. Peng, P. Abbeel, S. Levine, and M. van de Panne, "Deepmimic: Example-guided deep reinforcement learning of physics-based character skills," *ACM Trans. Graph.*, vol. 37, pp. 143:1–143:14, July 2018.

[14] H. Hasselt, "Double q-learning," in *Advances in Neural Information Processing Systems* (J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, eds.), vol. 23, Curran Associates, Inc., 2010.

[15] H. v. Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, p. 2094–2100, AAAI Press, 2016.

[16] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," 2015. cite arxiv:1511.05952Comment: Published at ICLR 2016.

[17] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling network architectures for deep reinforcement learning," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, p. 1995–2003, JMLR.org, 2016.

[18] M. Fortunato, M. G. Azar, B. Piot, J. Menick, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin, C. Blundell, and S. Legg, "Noisy networks for exploration," *CoRR*, vol. abs/1706.10295, 2017.

[19] M. Towers, J. K. Terry, A. Kwiatkowski, J. U. Balis, G. d. Cola, T. Deleu, M. Goulão, A. Kallinteris, A. KG, M. Krimmel, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, A. T. J. Shen, and O. G. Younis, "Gymnasium," Mar. 2023.