Learning to Strike Accurately with DQN-based Algorithms

Ayal Taitler and Nahum Shimkin

***Abstract*—**We consider in this paper the application of deep reinforcement learning techniques to learning closed loop control and goal-oriented trajectory planning in a robotic application. We employ an end-to-end (from the motor input the required task) model free approach using a deep Q-learning framework to learn a motoric skill. We propose several improvements to the naive deep Q-learning algorithm which otherwise fails. First we use some rough prior knowledge we have on the goal of the task to heuristically explore the environment. Second we manage to prevent the so-called catastrophic forgetting of neural networks. We present our simulation results for accurate striking task in air hockey, and show the success and stability of our learning algorithm due to the proposed modifications. We also present simulations that further support our claim of successfully mitigating the problem of catastrophic forgetting.

***Keywords*—**Air hockey, catastrophic forgetting, deep reinforcement learning, model free control, robot learning.

I. Introduction

THE problem of learning a skill, a mapping between states and actions to reach a goal in a continuous world, lies at the heart of every interaction of an autonomous system with its environment. In this paper, we consider the problem of a robot learning how to strike effectively the puck in a game of air hockey. Air hockey is a fast competitive game where two players face each other on a low-friction table. Players are required to develop and perfect skills such as blocking and striking in order to play and win. Classical approaches for striking the puck involve a hierarchical process of planning and execution. A common approach involves, first, planning a strategy based on the goal and skill, e.g., calculating the best point of collision to achieve the goal, then planning a path and trajectory, and finally executing the low level motoric control [23]. Each part requires full knowledge of the mechanical and physical models, which might be complex or unavailable. In this paper, we apply a model free reinforcement learning to a goal-oriented robotic task of striking a puck effectively towards a target point in an air hockey simulated environment. We propose doing the planning and the control simultaneously with end-to-end learning, which offers a model-free way to learn from the final result. Specifically, we employ a deep Q-learning approach, inspired by the spectacular success it had in the Atari 2600 video games [19]. The result will be given in a form of reward at the end of each strike attempt, and will serve as the reinforcement signal for the learning the correct striking policy.

When dealing with continuous robotic problems, policy gradients methods [39] are a popular approach, where a mapping between states and actions is learned with gradient ascent optimization on the accumulated reward, with or without keeping track of the value function. Another popular approach is Learning from Demonstration (LfD) [27], [3] sometimes refereed as imitation learning [21] and apprenticeship learning [2]. In LfD a human expert (or a programmed agent) is recorded and the learning agent learns on the recorded data in a supervised fashion. Sometimes this process is used as an initialization for a second reinforcement learning stage for improvement. In a similar application to ours [5] used imitation learning to learn primitive behaviors for a humanoid robot in air hockey.

The driving force of all reinforcement learning algorithms is the reward signal. The performance objective is also formulated as a maximization of the total reward. A learning agent evaluates each state and action according to the amount of reward that can be collected until reaching the goal state. When there are no rewards along the way, only at the end of an episode, it is hard for such an agent to distinguish between good and bad transitions. E.g., an episode which ended in a "bad" terminal state, might still contain good transitions. Thus, such problems have given rise to reward shaping and reward engineering techniques [17], [24] in order to guide the learning in cases otherwise prone for failure. This problem occurs especially when the state space is high dimensional, the action policy is complicated and the learning episode duration is long. Control and planning in robotics suffers from all of the above.

Exploration has been a standing problem since the early days of reinforcement learning and intelligent control [34]. The role and necessity of exploration have been discussed in length, and also how to perform the exploration efficiently [35]. Exploration has several aspects. The first is that the agent has to explore the environment to figure out on its own what is the required task based on the gathered rewards. Another aspect is once a policy is found that achieve some reward which is better than a random policy (that might be viewed as understanding the task), gradually improve the policy toward reward maximization. The ϵ-greedy exploration which is the most basic exploration method used, is not highly efficient in motor tasks, since a dynamical system functions as a low pass filter and once in a while using a random action might have little

This work was supported in part by a research grant from KLA-Tencor.

A. Taitler is with the Faculty of Electrical Engineering, Technion–Israel Institute of Technology, Haifa 32000, Israel (phone: +97248294739; email: ataitler@technion.ac.il).

N. Shimkin is with the Faculty of Electrical Engineering, Technion–Israel Institute of Technology, Haifa 32000, Israel (email: shimkin@technion.ac.il).

effect on the output of the system. We combined several types of explorations including integration of prior knowledge on the goal only as part of the learning process without any information on the robot's internal model or the problem structure, we also used ϵ-greedy and local exploration, for better exploration in systems with limited bandwidth operating in continuous spaces.

Deep neural networks are currently the most successful machine-learning tools for solving complex tasks. One weakness of such models is that, unlike humans, they are unable to learn continuously throughout complex tasks while keeping old information. The weights learned in the earlier stages of the task are being changed by later stages updates, causing for performance degradation in the states learned earlier. In this work we have dealt with this problem and practically prevented this degradation in performances completely.

We propose a deep Q-learning algorithm suitable for learning complex policies in dynamic physical environments. The algorithm combines ϵ-greedy exploration with a temporally correlated noise [16] for local exploration, which proved to be essential for effective learning. We further propose two novel contributions. We suggest a more relaxed approach to LfD which does not have the same limitations as standard LfD and can be learned from experience as regular RL as an exploration enhancing mechanism. We also manage to overcome the instability of the learning algorithm due to the non-stationarity of the observed data and the forgetting of old data, by gradually expending the target network update period. The target network is used to stabilize the learning and prevent oscillations as part of the deep Q-learning algorithm, and as we show affect the forgetting of the algorithm.

We compare our results with other deep reinforcement learning algorithms, namely Double DQN and Deep Deterministic Policy Gradients, and achieve significant improvements. We are able to reach near optimal performance, and keep the performance over time without suffering from a drop in the score function and the policies obtained. We also conduct experiments to understand the effect of our algorithm on the forgetting phenomenon.

II. Related Work

Related research on learning in robotics and autonomous systems was conducted in several directions. Traditionally reinforcement learning has been divided roughly into model-based learning and model-free learning. In model-based learning the learning agent estimates a dynamic model as part of the learning process, then exploit the estimated model to calculate the optimal policy [4], [32]. In model-free learning the agent tries to estimate an effective policy, without learning an explicit model, either by learning it directly [39] or estimating the value function [33]. Popular methods in model free learning for learning the value function calculate the temporal difference (TD) error for the updates. The TD can be used in on-policy algorithms such as TD(0) and Sarsa, which learn the value function of a given policy, and gradually attempt to improve the policy. The TD can also be used in off-policy algorithms which learns from samples in order to learn directly the optimal value function [38] for deriving the optimal policy directly.

Policy learning methods for episodic (finite duration) problems usually estimate the policy directly from full episode roll-outs by using stochastic policies. Policy gradient methods have evolved in several directions, including natural gradients and actor-critic methods, where the last attempts to keep track of the value function for better policy updates. In recent years a class of deterministic policy gradients have also been developed [31]. Policy gradients were very successful in many domains but struggled as the number of parameters increased. For a comprehensive survey see [14], [8].

Since the groundbreaking results shown by Deep Q-Learning [19] for learning to play computer games on the Atari 2600 arcade environment such as Breakout and Demon Attack, there has been extensive research on deep reinforcement learning. Deep Q-learning in particular seeks to approximate the Q-values using deep networks, such as deep convolutional neural networks. There has been work on better target (see Sec. III) estimation [37], improving the learning by prioritizing the experience replay buffer to maximize learning [28] and preforming better gradient updates with parallel batch approaches [18], [22]. Policy gradient methods have also enjoyed the success of deep neural networks, and several approached have been introduced such as [15] which learns with the help of DDP generated trajectories, and more recently [29] which attempt to keep track of the advantage function instead of the commonly used Q-function. An attempt to combine between the deterministic policy gradients with deep neural networks and the ideas presented in the DQN algorithm have produced the deep deterministic policy gradients [16], which will be used for comparison in this work. Several benchmark studies such as [9] have made comparisons between continuous control algorithms.

Our work has been influenced mainly by the recent work on deep Q-networks [19], [20], [30], [37], and the adaptation for continuous domains of deep deterministic policy gradients [16]. In This paper we focus on the on-line DQN-based approach, and extend it to the domain of continuous state optimal control for striking in air hockey.

Imitation learning is a popular approach when concerning robotics and dynamical systems in general. In application related to ours, a humanoid was taught to play air hockey from expert's recordings [5]. In [21] a robot playing table tennis was taught how to play using motor primitives. A recent work used deep reinforcement learning with imitation learning [21] for learning control policies in Atari and simple control problems, and have successfully transitioned between the imitation stage and the on-line stage where the agent continues to learn on its own.

In the on-line scheme of learning the exploration is a critical part. In high dimensional continuous spaces often methods involving optimism or directed exploration are used. Often is the case that the environment does not supply an interacting agent with continuous flow of rewards. The scarcity of rewards affects the ability of the agent to efficiently explore the environment. A popular approach is to augment external received rewards with internal generated rewards [7]. Recently a curiosity-based, approach where the agent rewards itself based on a curiosity measure [26] has gained popularity. In this case the reward maximization is done over the total accumulated rewards of internal and external received rewards.

Catastrophic Forgetting (CF) [10] is a problem observed in neural networks when learning in high dimensional continuous spaces, such as in the case of control of dynamical systems. Paper [11] have proposed a neuron selection technique which keeps local representation of the value function. When learning several tasks, older tasks tend to be forgotten. In order to address that [13] have proposed a regularization term which penalized deviations of the neural network weights from the previous learned ones.

III. Deep Q-Networks

We consider a standard reinforcement learning setup consisting of an agent interacting with the environment in discrete time steps. At each step the agent receives an observation $s_t \in \mathbb{R}^n$ which represents the current physical state of the system, takes a discrete action $a_t \in A$ which it applies to the environment, receives a scalar reward $r_t = r(s_t, a_t)$, and observes a new state s_{t+1} which the environment transitions to. It is assumed that the next state is according to a stochastic transition model $P(s_{t+1}|s_t, a_t)$. The action set $\mathcal{A}$ is assumed to be discrete.

The goal of the agent is to maximize the sum of rewards gained from interaction with the environment. Our problem is a finite horizon problem in which the game terminates if the agent reached some predefined time T. We define the future return at time t as $R_t = \sum_{t'=t}^{T} r_{t'}$, where T is the time at which the game terminates. The goal is to learn a policy which maximizes the expected return $\mathbb{E}[R_0]$ from the initial state.

The action-value function $Q^*(s, a)$ is used in many reinforcement learning algorithms. It describes the expected return after taking an action a in state s and thereafter following an optimal policy. The optimal state-action value function Q^* obeys the equality known as the *Bellman's equation*:

$$Q*\left(s_t, a_t\right) = \mathbb{E}\left[r_t + \max_{a'}\left(s_{t+1}, a'\right) \middle| s_t, a_t\right] \tag{1}$$

For learning purposes it is common to approximate the value of $Q^*(s, a)$ by using a function approximator, such as a neural network. We refer to the neural network function approximator with weights θ as a Q-network. A neural network representing the Q-function can be trained by considering the loss function:

$$L(\theta) = \mathbb{E}_{s_t, a_t, r_t, s_{t+1} \sim D}\left[\left(y(\theta) - Q\left(s_t, a_t; \theta\right)\right)^2\right] \tag{2}$$

Where

$$\begin{array}{ll} y(\theta) = r\left(s_t, a_t\right) & s_{t+1} \text{ is terminal} \\ y(\theta) = r\left(s_t, a_t\right) + \max\limits_{a} Q\left(s_{t+1}, a; \theta\right) & \text{otherwise} \end{array} \tag{3}$$

During training each transition of state, action, reward and next state $< s_t, a_t, r_t, s_{t+1} >$ is stored in an *experience replay buffer* D from which samples are drawn uniformly in order to reduce time correlations to train the network. $y(\theta)$ is called the target value and is also a function of θ. The max operator in the target makes it hard to calculate derivatives with respect to the weights, so the target is kept constant and the derivatives are calculated only with respect to $Q(s_t, a_t; \theta)$. This loss function has the tendency to oscillate and diverge. In order to keep the target stationary and prevent oscillations, the DQN algorithm makes use of another network, called a target network with parameters $\hat{\theta}^-$. The target network is the same as the on-line network except that its parameters are copied every C updates from the on-line network, so that $\hat{\theta}^-$. are kept fixed during all other updates. The training of the network in this case is according to the following sequence of loss functions:

$$L_i\left(\theta_i\right) = \mathbb{E}_{s_t, a_t, r_t, s_{t+1} \sim D}\left\lfloor\left(y\left(\hat{\theta}_i^-\right) - Q\left(s_t, a_t; \theta_i\right)\right)^2\right\rfloor \tag{4}$$

The target used by DQN is then

$$y_i\left(\hat{\theta}_i^-\right) = r\left(s_t, a_t\right) + \max_{a} Q\left(s_{t+1}, a; \hat{\theta}_i^-\right) \tag{5}$$

And the on-line network weights can be trained by stochastic gradient descent (SGD) and back-propagation

$$\theta_{i+1} = \theta_i + \alpha \nabla_{\theta_i} L_i\left(\theta_i\right) \tag{6}$$

Where α is the learning rate. An improvement on that has been proposed in the Double DQN (DDQN) algorithm. The estimation of the next value, and the selection of the action, have been decoupled. This separation decreases the problem of value overestimation, thus the following target has been used:

$$\begin{aligned} y_i\left(\hat{\theta}_i^-\right) &= r\left(s_t, a_t\right) + Q\left(s_{t+1}, a_{t+1}; \hat{\theta}_i^-\right) \\ a_{t+1} &= \arg\max_{a} Q\left(s_{t+1}, a; \hat{\theta}_i^-\right) \end{aligned} \tag{7}$$

In our work unless specified otherwise all learning updates have been done according to the double DQN learning rule.

To explore the environment, the basic ϵ-greedy exploration scheme is used. Given a state, a deep Q-network (DQN) predicts a value $\hat{Q}(s, a)$ for each action a. The agent chooses the action with the highest value with probability $1 - \epsilon$ and a random action with probability ϵ.

IV. Striking in Air Hockey

We next introduce the striking problem and our learning approach.

A. The Striking Problem

The striking problem deals in general with interception of a moving puck and striking it in a controlled manner. We specialize here to the case where the puck is stationary. We wish to learn the control policy for striking the puck such that after the impact, the puck trajectory will have some desired properties. We focus on learning to strike the puck directly to the opponent's goal. We also considered some other different modes of striking the puck, Such as hitting the wall first. These are not presented here, but the same learning scheme fits them as well. We refer to these modes as skills, which a high level agent can choose from in full air hockey game. The learning goal is to be able to learn these skills with the following desired properties

- The puck's velocity should be maximal after the impact with the agent.
- The puck's end position at the opponent's side should be the center of the goal.
- The puck's direction should be according to the selected skill.

The agent is a planar robot with 2 degrees of freedom, X and Y (gantry like robot). We used a second order kinematics for the agent and puck. The state vector of the problem is $s_t \in \mathbb{R}^8$, which includes all the position and velocities of the agent and the puck in both axes, i.e., $s_t = \left[m_x, m_{Vx}, m_y, m_{Vy}, p_x, p_{Vx}, p_y, p_{Vy}\right]^T$. Here m_* stands for the agent's state variables and p_* stands for the puck's state variables. The actions are $a_t \in \mathbb{R}^2$, and include the accelerations in both axes for the agent.

The striking problem can be described as the following discrete time optimal planning problem:

$$\begin{aligned} \underset{a_k}{\text{minimize}} \quad & \phi\left(s_T, T\right) \\ \text{subject to} \quad & s_{k+1} = f\left(s_k, a_k\right) \\ & s_k^{(i)} \in \left[S_{\min}^i,\ S_{\max}^i\right] \qquad i = 1, \ldots, 8 \\ & a_k^{(j)} \in \left[A_{\min}^j,\ A_{\max}^j\right] \qquad j = 1, 2 \\ & s_0 = s(0) \end{aligned} \tag{8}$$

Here the objective function $\phi(s_T, T)$ represents the value of the final state s_T (in terms of velocity and accuracy), and the final time T which we desire to be small. The function $f(\cdot)$ is the physical model dynamics. $S_{min}^{(i)}$, $S_{\max}^{(i)}$ and $A_{\min}^{(j)}$, $A_{max}^{(j)}$ are the constraints on the state (table boundaries and velocities) and action spaces (accelerations/torques) respectively. s_0 is the initial state. We assume that $f(\cdot)$, the collision models and the table state constraints are hidden from the learning algorithm, The best known collision model is non-linear and hard to work with [25]. Solving analytically such a problem when these function are known is a challenging problem, when they are unknown it is practically impossible with analytic tools. In the simulations specific models were specified as explained in Section V.

In order to fit the problem as stated in IV-A to the DQN learning scheme, where the outputs are discrete Q values associated with discrete actions, we discretized the action space by sampling a $2D$ grid with n actions in each dimension (each dimension represents an axis in joint frame). Thus, we have n^2 actions. We make sure to include the marginal and the zero action, so our class of policies we search in will include the Bang-Zero-Bang profile which is associated with time optimal problems. Each action is associated with an output of the neural network, where each output represents the Q-values of each action under the input state supplied to the network, e.g., if state s is supplied to the network, output i is the Q-value of $Q(s, a_t; \theta)$. Thus, for every given state we have n^2 Q-values from the network, associated with the n^2 actions.

B. Reward Definition

The learning is episodic and in this problem the agent receives success indication only upon reaching a terminal state and finishing an episode. The terminal states are states in which the mallet collide with one of the walls (table boundaries violation), and the states in which the mallet strikes the puck (the agent does not perform any actions beyond this point). Any other state including the states in which an episode terminates due to reaching the maximal allowed steps, are not defined as terminal states. At the terminal state of each episode the agent receive the following reward:

$$R_{\text{terminal}} = r_c + r_v + r_d \tag{9}$$

$R_{terminal}$ consists of three components. The first is r_c, which is a fixed reward indicating a puck striking. The second component is a reward which encourages the agent to strike the puck with maximum velocity, and given by

$$r_v = sign\left(V\right) \cdot V^2 \tag{10}$$

Where V is the projection of the velocity on the direction of a desired location x_g on the goal line of the opponent. The last component is a reward for striking accuracy, which indicates how close the puck reached x_g.

$$r_d = \begin{cases} c & \left|x - x_g\right| \le w \\ c \cdot e^{-d\cdot\left(\left|x-x_g\right|-w\right)} & \left|x - x_g\right| > w \end{cases} \tag{11}$$

Where x is the actual point the puck reaches on the opponent's side on the goal line, c is a scaling factor for the reward, w is the width of the window around the target point which receives the highest reward and d is a decay rate around the desired target location. Naturally, if the episode terminates without striking the puck $R_{terminal}$ is zero. In order to encourage the agent to reach a terminal state in minimum time, the agent receives a negative small reward $-r_{time}$ for each time step of the simulation until termination. The accumulative reward for the entire episode then is $R_{total} = R_{terminal} - n \cdot r_{time}$, where n is the number of time steps for that episode.

C. Exploration

The problem of Exploration is a major one, especially in the continuous domain. We address the issue from two angles, completely random exploration and local exploration.

1) Completely Random Exploration:

We use ϵ-greedy exploration (see Section III) in order to allow experimenting with arbitrary actions. In physical systems with inertia it is not efficient since the system acts as a low pass filter, but it does give the agent some sampling of actions it would not try under normal conditions.

2) Local Exploration

The main type of exploration is what we refer to as *local* exploration. Similarly to what was done in [16], we added a noise sampled from a noise process $\mathcal{N}$ to our currently learned policy. Since the agent can apply only actions from a discrete set of actions $\mathcal{A}$, we projected the outcome on the agent's action set:

$$a_t = \mathcal{P}_{\mathcal{A}}\left\{\arg\max_a Q\left(s_t, a; \theta\right) + \mathcal{N}_t\right\} \tag{12}$$

We used for $\mathcal{N}$ an Ornstein-Uhlenbeck process [36] to generate temporally correlated exploration noise for exploring efficiently. The noise parameters should be chosen in such a way that after the projection the exploration will be effective. Small noise might not change the action after the projection, but large noise might result in straying too far from the greedy policy. Thus, the parameters of the noise should be in proportion to the actions range and the aggregation.

D. Prior Knowledge as Exploration Guiding Heuristic

In a complex environment, learning from scratch has been shown to be a hard task. Searching in a continuous high dimensional spaces with local exploration might prove futile. In many learning problems prior knowledge and understanding are present and can be used to improve the learning performance. A common way of inserting priors into the learning process uses LfD. For that purpose, multiple samples of expert performance should be collected, which is not always feasible or applicable.

In many cases the prior knowledge can be translated to some reasonable actions, although usually not an optimal policy.

Examples for that can be seen in almost every planning problem. In games, the rules give us some guidance to what to do, e.g., in soccer, "Kick the ball to the goal," so for an agent to spend time on learning the fact that it has to kick the ball is a waste. In skydiving, the skydivers are told to move their hands to the sides in order to rotate, they are not required to search every possible pose to learn how to rotate. Furthermore, the basic rotating procedure taught to new skydivers is not the correct way to do it, it is taught as a basic technique, an initialization for them to modify and find the correct way.

We propose showing the agent a translation of the prior knowledge as a heuristic teacher policy which in only aware on the goal of the task, and has no knowledge of the models involved. In some episodes instead of letting the agent to act according to the greedy policy, it does what the teacher policy suggests. The samples collected in those episodes are stored in the experience replay buffer as any other samples, allowing the learning algorithm to call upon that experience from the replay buffer and learn from it in within the standard framework.

For the problem of air hockey, we used a policy encapsulating some very crude knowledge we have of the problem's goal. We just instruct the agent to move in the direction of the puck, regardless of the task at hand (aiming to the right\left\middle) and the dynamic model, since this knowledge was simple, and robust enough. The guidance policy we constructed has the following form:

$$\begin{aligned} V_{next} &= \frac{P_{puck} - P_{agent}}{\left\| P_{puck} - P_{agent} \right\|} \cdot \mathrm{maxVelocity} \\ \mathrm{a} &= \frac{\frac{V_{next} - V_{agent}}{\Delta T}}{\left\| \frac{V_{next} - V_{agent}}{\Delta T} \right\|} \cdot \mathrm{maxForce} \end{aligned} \tag{13}$$

Where P_{object} is the x, y position vector of the object, and MaxVelocity, MaxForce are physical constraints of the mechanics. The agent acts by the projection of the policy on its action space $\mathcal{P}_{\mathcal{A}}\{\cdot\}$ This policy will result with an impact between the agent and puck, but by no account will be considered as a "good" strike since there is no reason the puck will move in the goal's direction (except in the special case when the puck lays on the line between the agent and the goal). The guidance policy is shown (the agent acts by it) and stored in the replay buffer with probability ϵ_p.

E. Non-Uniform Target Network Update Periods

The deep reinforcement learning problem is different from the supervised learning problem in a fundamental way, as the data which the network uses during the learning changes over time. At the beginning, the experience replay buffer is empty, the agent starts to act and fills the buffer, when the buffer reaches its maximal capacity new transitions overwrite the older ones. It is obvious that the data is changing over time, first changing in size and then changing in distribution. As the agent learns and gets better, the data in the buffer reflects that and the samples are more and more of good states which maximize the reward.

Recall that the value the neural network tries to minimize is the loss function stated in (2). In order to stabilize the oscillations a target network with fixed weights over constant update periods were introduced. That led to the stationarity of the target value. The choosing of update period length became of the parameters that had to be set. Small update period result with instability since the target network changes too fast and oscillates, large update periods may be too stationary and the bootstrap process might not work properly. Thus, a period that is somewhere in the middle is chosen so the updates are stable.

In many domains such as in the air hockey and also in some of Atari games, DQN still suffers from a drop in the score as the learning process progresses (see, e.g., Fig. 2). We argue that this drop is not only due to value overestimation (it happens for Double DQN updates as well), but also for issues with the target value. Choosing a middle value for the update period may result in slow learning in the beginning and a drop in the score later in the learning due to oscillations.

We show that by adjusting the update period over time, we manage to stabilize the learning and prevent completely the drop in the score. We start with a small update period since the replay buffer D is empty and we want to learn quickly, we then keep expanding the period as the buffer gets larger, and we need more sampling to cover it. As the agent gets better and the distribution stabilizes, we also expand the update period in order to filter oscillations and keep the agent in the vicinity of the good learned policy. The expansion of the update period is done at each target weights update according to

$$C = C \cdot C_r, \qquad C_r \geq 1 \tag{14}$$

Where C_r is the expansion rate. When $C_r = 1$ the updates are uniform as in the standard DQN.

At the beginning every sample contains new information that should affect the learning. As the learning progresses and the optimal policy hasn't been obtained yet, the samples in the replay buffer are diverse allowing the agent to learn from good samples and bad samples as well. At later stages when the agent has already learned a good policy, and the distribution of samples in the replay buffer resembles that policy. The network at the point if learning continuous, might suffer from what is known as the catastrophic forgetting [10] of neural networks. Freezing the target network before that stage, stabilize the learning and allows the network fine tune its performances, even though the distribution in the replay buffer is undiverse. The target network contains then the knowledge gained in the past from bad examples. At that stage of the learning the update period should be large for that purpose. This is achieved by gradually increasing the update period from an initial small period at the beginning during the learning.

F. Guided-DQN

Putting the above-discussed features together produces the guided-DQN algorithm we used in the air hockey problem. The algorithm is given in algorithm 1.

As an input the algorithm gets the guidance policy, which encapsulated the prior knowledge we have on the problem, and will be used as a heuristic guidance. The algorithm should also be supplied with the expansion rate C_r which controls the expansion rate of the target update periods. Before the algorithm begins the finite capacity replay buffer is initialized to be empty, the on-line neural network is initialized with random uniformly distributed values in each layer according the fan-in of the layer. The values of the target neural networks are copied from the on-line network so at the beginning the networks are identical.

The algorithm has several parameters which define its operation. The first is the number of episodes or striking tries M the algorithm will perform. A basic update rate C for the target network, a very small number since it is expending over the course of the algorithm. A learning rate α for the gradient updates. A batch size N for the sampling of transitions from the replay buffer. And finally two probabilities which define the exploration mechanism, ϵ_p is the rate in which a full guided episode is performed, and ϵ which is the rate in which the ϵ-greedy exploration chooses an action at random.

Algorithm 1: Guided Deep Q-Network

```
input : Guidance policy $\pi(s)$, expansion rate $C_r$
Initialize replay memory $D$
Initialize states-actions value function $Q$ with random weights $\theta$
Initialize target states-actions value function $\hat{Q}$ with weights $\hat{\theta}^{-} = \theta$
for $episode = 1, M$ do
    Observe initial state from environment $s_0$
    Initialize random process $n$ for action exploration
    with probability $\epsilon_p$ decide if this episode is guided or not
    while $t < N$ and $s_t$ is not terminal do
        if guided episode then
            Select $\mathcal{P}_{\mathcal{A}}\left\{\underset{a}{\operatorname{argmax}}\ \pi(s_t)\right\}$
        else
            With probability $\epsilon$ select random action $a_t$
             otherwise select
            $\mathcal{P}_{\mathcal{A}}\left\{\underset{a}{\operatorname{argmax}}\ Q(s_t, a) + n_t\right\}$
        Execute action $a_t$ in environment and observe reward $r_t$, next state $x_{t+1}$ and if terminal $d_{t+1}$
        Set new state $s_{(t+1)} = s_t,\ x_{t+1}$
        Store transition $< s_t, a_t, r_t, s_{t+1} >$ in $D$
        Sample random mini-batch of transition $< s_j, a_j, r_j, s_{j+1} >$ from $D$
        Set $y_t = r_t + \hat{Q}(s_{j+1}, \underset{a}{\operatorname{argmax}} Q(s_{j+1}, a; \theta); \hat{\theta}^{-})$
        Perform gradient descent step on $\left(y_j - Q(s_j, a_j; \theta)\right)^2$ with respect to the network weights $\theta$
        Every $C$ steps reset $\hat{\theta} = \theta$, and set $C = C \cdot C_r$
return Q
```

At each episode, with probability ϵ_p the entire episode will be executed with the guidance policy $\pi(s)$, or with probability $1 - \epsilon_p$ according to the greedy policy, in case of a greedy policy episode the actions are augmented with a time correlated exploration noise for the local exploration. In either case, a guided episode or a greedy episode, at each step the algorithm executes the selected action, observes the new transitioned state and reward, and stores the transitions in the replay buffer. Then a batch of samples from the replay buffer are selected randomly with uniform probability, and a learning step according the Double DDQN loss and gradient descent is performed on the on-line Q network. Every C updates the target Q network is updated with the weights of the on-line Q network, and C is expanded with a factor of C_r so the next time the target network gets updated, it will be after a longer period than the previous update. The algorithm ends after M episodes.

The projection operator $\mathcal{P}_{\mathcal{A}}$ projects the continuous actions onto the agent's discrete set, by choosing the action with the lowest Euclidean distance. The learning rule is a Double DQN learning rule. Note that if the algorithm is not provided with a guidance policy (equivalent to setting ϵ_p to zero), $C_r = 1$, and the temporal correlated process is $\mathcal{N} \equiv 0$, the GDQN algorithm reduces to the standard Double DQN algorithm.

V. Experiments

The simulation was fashioned after the robotic system in Fig. 1.

In the robotic system the algorithm would learn on the real unknown physical models, but for the purpose of simulation we used simulation models for the agent dynamics and collision models. The simulation models are hidden from the learning algorithm and exist solely for the purpose of simulating the system for learning. For the agent dynamics we used a discrete time second order dynamics

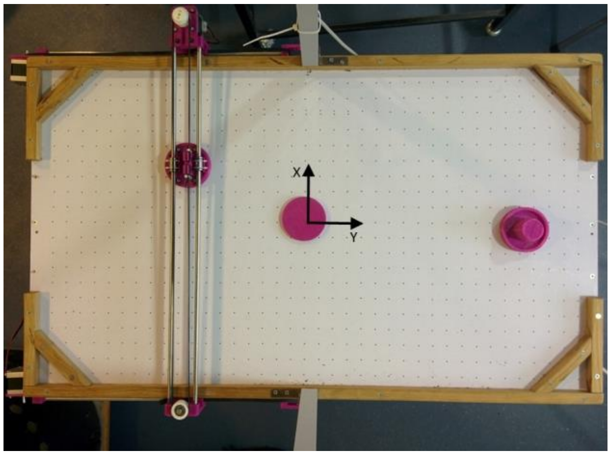

Fig. 1 mapping nonlinear data to a higher dimensional feature space.

$$\begin{bmatrix} X_m \\ V_{x,m} \\ Y_m \\ V_{y,m} \end{bmatrix}_{k+1} = \begin{bmatrix} 1 & T & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & T \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_m \\ V_{x,m} \\ Y_m \\ V_{y,m} \end{bmatrix}_k + \begin{bmatrix} 0 & 0 \\ T & 0 \\ 0 & 0 \\ 0 & T \end{bmatrix} \begin{bmatrix} a_x \\ a_y \end{bmatrix}_k \tag{15}$$

Under the following constraints

$$\left|a_{x,y}\right| < \text{Maximum force}$$
$$\left|V_{\{x,y\},m}\right| < \text{Maximum velocity}$$
$$\left|X_m, Y_m\right| < \text{Table bounderies}$$

These constraints represent the physical constraints present in the mechanical system, where the velocity has a maximum value, the torques are bounded and we are not allowing the mallet to move outside of the table boundaries.

We used in the simulations an ideal impact model between the mallet and puck in the sense that we neglected the friction between the bodies during the impact and we assume the impact is instantaneous with energy loss according to a restitution coefficient e. The forces, accelerations, velocities and space (the field's boundaries) are constrained to reflect the physical constraints in the robotic system.

The list of parameters (learning rate, probabilities, etc.) used throughout the simulations is given in table I.

The learning environment is a custom built simulation based on OpenAI gym [6]. The simulation is modeled after an air hockey robotic system with two motors and track, one for each axis. The simulation includes visually the table, the mallet and the puck. The environment and learning scripts were written in python, and TensorFlow [1] for the purpose of models training. The simulations were conducted on a computer with a single NVIDIA Titan X GPU and 65GB of RAM. Each simulation (100K episodes) took approximately twelve hours.

TABLE I
LIST OF HYPER-PARAMETERS USED BY THE LEARNING ALGORITHM

Parameter	Value	Description
α	0.00025	Learning rate
ϵ	0.1	ϵ-greedy exploration probability
ϵ_p	0.1	Policy demonstration probability
D	$2 \cdot 10^5$	Replay buffer size
N	300	Episode's max steps
$\mathcal{A}$	5×5	Action space size
Mini-Batch	64	Mini-batch size sampled from buffer
C	Expanding	Update rate of Q target network
C_r	1.2	Expansion rate

We simulated each attempt to strike the puck as an independent episode comprised of discrete time steps. At the beginning of each episode the puck is placed at a random position on the table at the agent's half court with zero velocity and the agent starts from a home position (a fixed position near the middle of the goal) with zero velocity. Each episode terminates upon reaching a terminal state or upon passing the maximum number of steps defined for an episode. The maximum steps number is 150 steps and the terminal states are the states where the agent collides with the puck ("good" states) or with one of the walls ("bad" states). The environment returns a reward as described in Section IV-B. No reward is given upon hitting a wall beyond the timely reward.

The dynamic model of the puck and agent is a second order model as described in Section IV-A. T is the sampling time of the system and was set to 0.05 $[sec]$ in the simulation. The puck's rotation was neglected, thus the collision models (puck-agent, puck-wall) are ideal with inbound and outbound angles the same. Energy loss in the collisions was modeled with restitution coefficient of $e = 0.99$.

The controller is a non-linear neural controller, a fully connected Multi-Layer Perceptron with 4 layers (3 hidden layers and an output layer), the first two hidden layers are of 100 units each, the third hidden layer is of 40 units and the output layer is of 25 units. All activations are the linear rectifier $f(x) = max(0, x)$. The controller is a map between states s_t (the inputs to the controller) and discretized Q-values. We choose 5 actions in each axis, yielding 25 output actions\Q-values (see Section IV-A). We used the RMSProp algorithm with mini-batches of size 64.

In all the simulation experiments we measured the score for random initial positions of the puck, it will always be shown in graph with the caption random. In addition we measured the performances for additional 3 fix representing states of the puck, fixed positions in the left side, the middle and the right side of the table. In addition we estimated the average value of all the states and present it as well. The graphs matching these measures will be shown with appropriate captions. We present in this paper the results for the "direct hit."

A. Striking Results

First we show the performance of the standard Double DQN in Fig. 2 for different target network update periods. We choose a fast period an intermediate period and a slow period calculated such that each state in the buffer will be visited 8 times on average before being thrown away from the buffer.

It can be seen that the Double DQN with fast updates ($DDQN_{200}$) rises the fastest but also drops quickly, the same behavior can be observed for the intermediate updates ($DDQN_{1000}$) but the rise is slower and the drop happens less sharply. The score value the network drops to, -150, is exactly the value of the time penalty for a complete episode, i.e., the agent doesn't reach a terminal state. When investigating the policies obtained it can be seen that the agent's action oscillated between two opposite actions which affectively cause it to stand still. For the slow updates ($DDQN_{5000}$) the case is different, the network seems mostly indifferent to the updates, and at the end it manages to rise a little. The average value for all three runs oscillates and in general suffers from severe underestimation.

In Fig. 3 we compare the results of three algorithms, the Double DQN algorithm with the intermediate update period (the best of the three shown before), Deep-mind's Deep Deterministic Policy Gradients (DDPG) algorithm, and our Guided-DQN algorithm.

The DDPG algorithm manages to learn a suboptimal policy, but oscillates strongly around it. It can be seen in the fix positions graphs of the puck, although in the random graphs it looks pretty stable on the suboptimal policy. Double DQN was discussed before, and our GDQN as can be seen clearly, learns the optimal policy and reaches the maximum score possible for each of them. In the random puck position the score also reaches an optimal policy in a very stable manner. Note that the score doesn't drop at all, and even the rise at the beginning is faster than the other two algorithms, it even faster than the rise of the Double DQN with the fast updates shown in Fig. 2, due to the fast updates at the beginning and the guidance of the teacher policy. The average values of Double DQN and DDPG are oscillating and suffering from underestimation and overestimation respectively, where GDQN's average value is extremely stable and does not suffer from over or under estimation.

We measured the control signal and the trajectories for each of the three fixed positions of the puck. In Fig. 4 we see the results for a puck stationed in the left side of the table and in

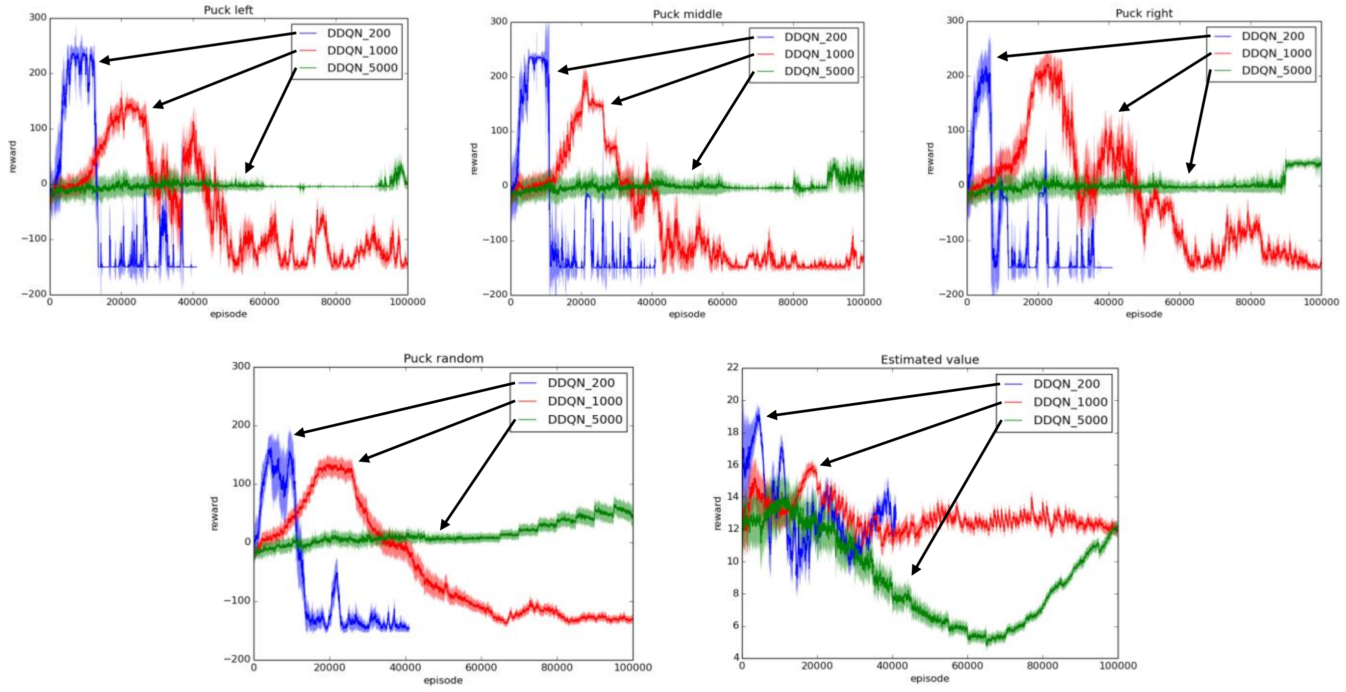

Fig. 2 Double DQN results for the air hockey striking problem.

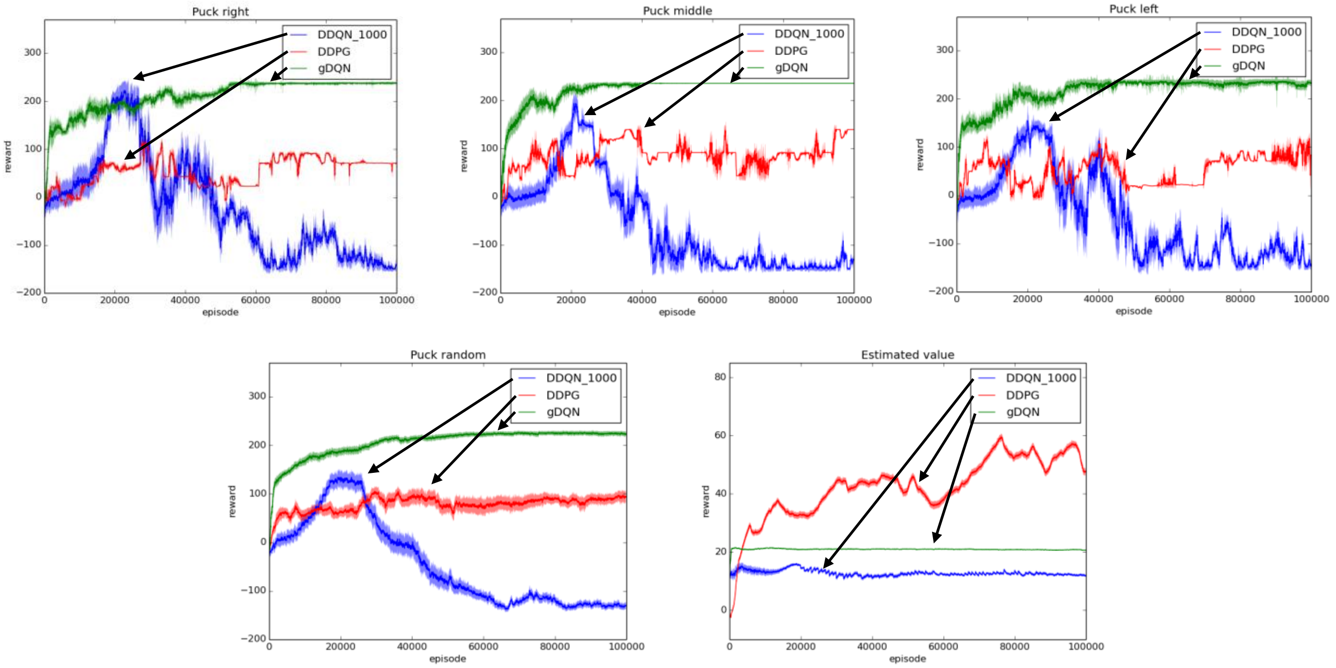

Fig. 3 GDQN, DDQN and DDPG results for the air hockey striking problem.

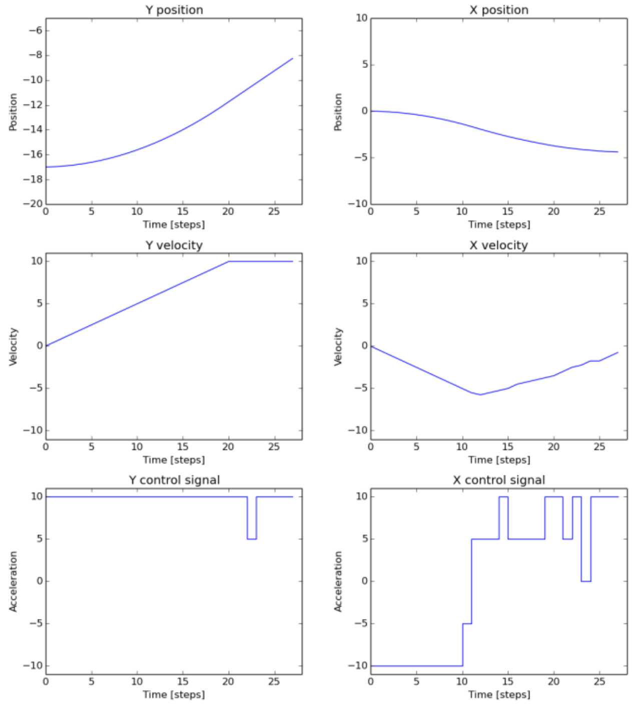

Fig. 4 Learned profiles for a puck stationed in the left side of the table. The rows are positions velocities and control signal send to the agent respectively.

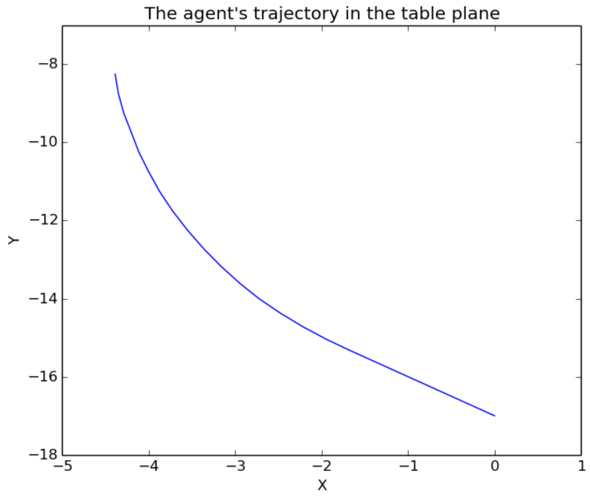

Fig. 5 The agent's trajectory in the X-Y plane of the table, for a left stationed puck.

Fig. 5 we can see the profile in the X-Y plane. In Fig. 6 and Fig. 7 the profiles and trajectories for a puck stationed in the middle of the table can be seen, and in Fig. 8 and Fig. 9 the same for a puck stationed in the right of the table. In all cases the X axis is the horizontal dimension of the table, and Y axis the vertical dimension of the table.

In all cases the agent's trajectory is following the expected curves in order to hit the puck so it will go to the middle of the goal. The motion is visually very similar to an S-curve. The control signal is one of the axes is saturated, and in the other is either saturated or zero (or small oscillations around zero) as necessary, in compatibility with the Band-Zero-Bang profile.

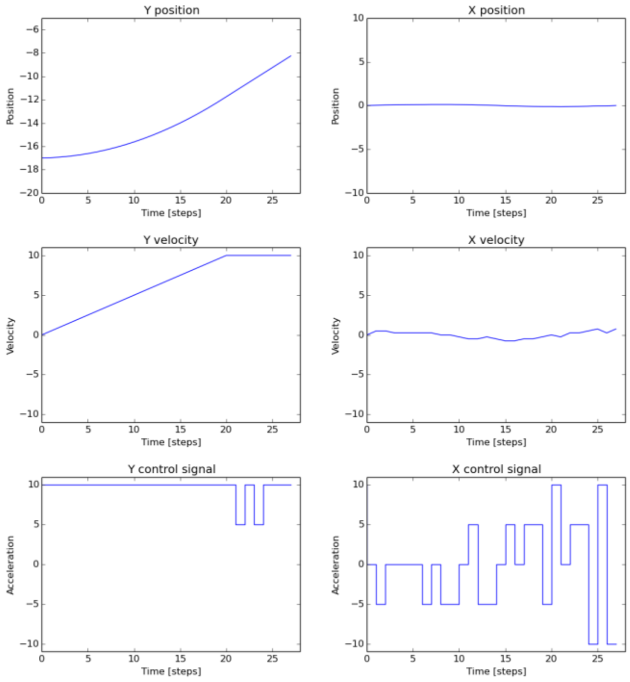

Fig. 6 Learned profiles for a puck stationed in the middle of the table. The rows are positions velocities and control signal send to the agent respectively.

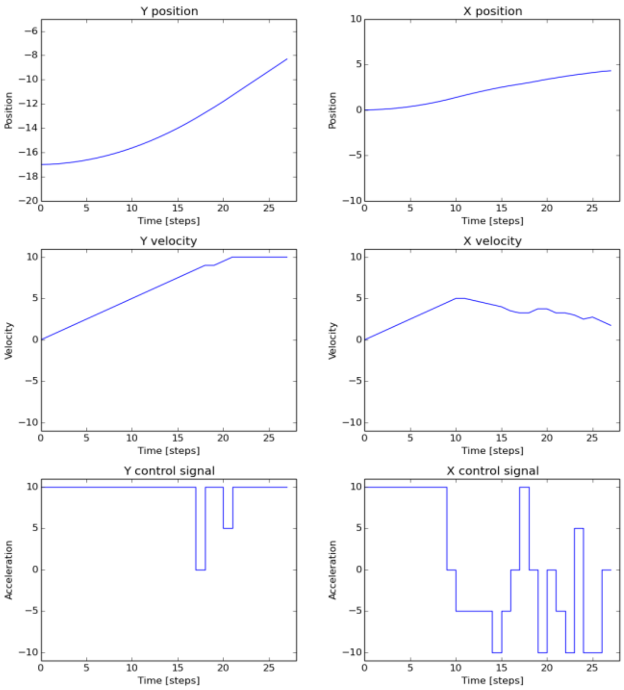

Fig. 8 Learned profiles for a puck stationed in the right side of the table. The rows are positions velocities and control signal send to the agent respectively.

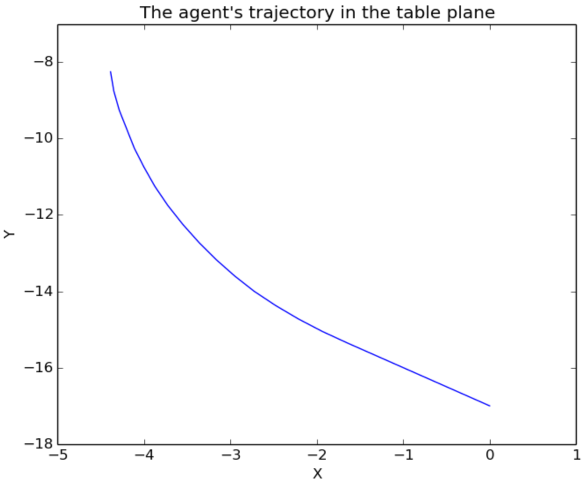

Fig. 7 The agent's trajectory in the X-Y plane of the table, for a middle stationed puck.

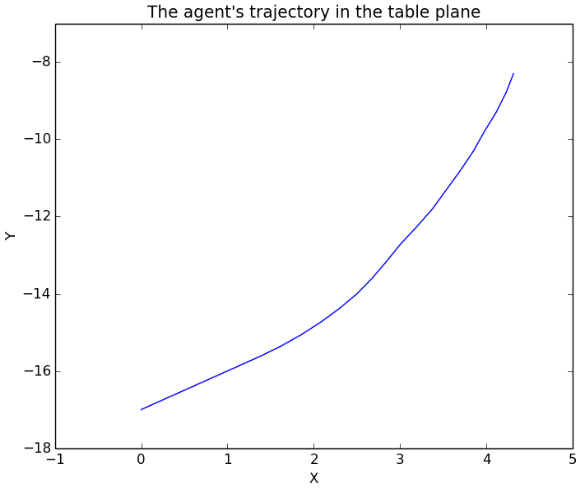

Fig. 9 The agent's trajectory in the X-Y plane of the table, for a right stationed puck.

B. Catastrophic Forgetting Analysis

Here we attempt to establish the claim that our learning network suffers from the Catastrophic Forgetting (CF) phenomena. In order to understand the drop in the score function which we associate with CF, we conducted two experiments. In the first experiment we learned with Double DQN just to the point where the learning reaches a plateau (8000 episodes), and then freezed the experience replay buffer, so the transitions stored to this point will not be overwritten, and then allowed the algorithm to continue its course (the target network is updated but the replay buffer is fixed). In the second experiment we did the same up to the point of the plateau, but instead of freezing the replay buffer we froze the target network, so the new experienced transitions will erase old transitions in the buffer, but the target network will not be copied any more. In Fig. 10 we show the results for these two

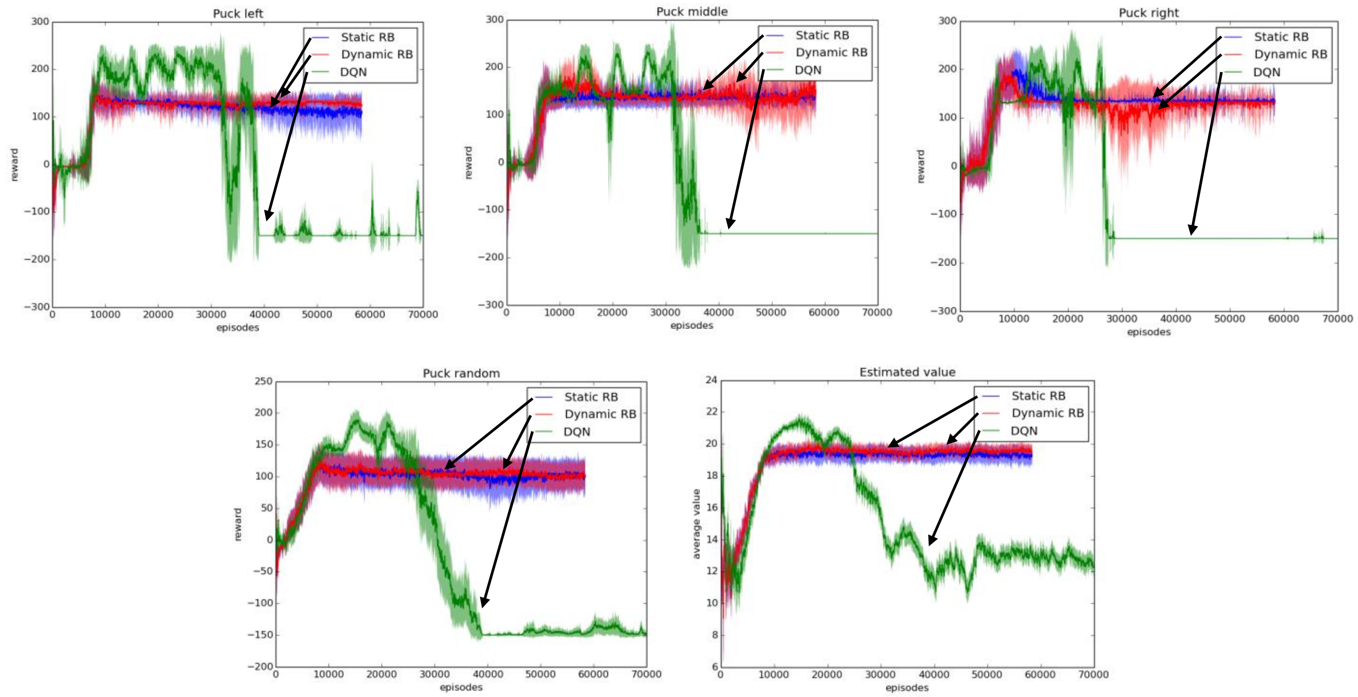

Fig. 10 DDQN with static replay buffer, DDQN with static target network and standard DDQN results for the air hockey striking problem.

experiments with the results of the Double DQN algorithm for reference. We can see clearly that both of the experiments did not suffer from degradation in the performance (nor any improvement as expected). From the first experiment we can understand that the transitions not seen any more indeed affect our learning. The old transitions were not overwritten so the algorithm had used them for the learning, and the performances reflected the agent's ability to exploit its current knowledge, and so the score remained the same. In the second experiment we allowed for experience overwriting but kept the target network fixed, again the results were similar to the first experiment, which lead us to associate the role of active memory representation to the target network.

VI. Concluding Remarks

We addressed the application of striking a stationary puck with a physical mechanism. This application proved challenging for the standard Double DQN algorithm. Therefore we proposed two novel improvements to this algorithm.

1. Using prior knowledge during learning to direct the algorithm to interesting regions of the state and action spaces.
2. Using non-uniform target update periods with increasing duration in order to stabilize the learning process.
3. Augmenting the plain ϵ-greedy exploration mechanism with a local exploration with temporally correlated random process to better accommodate for the physical environment.

The modified algorithm is shown to learn near optimal performance in the motion planning and control problem of air hockey striking. In particular, it solves completely the problem of score drop that was observed in Double DQN.

While considering the problem of learning a striking policy in air hockey, we came across a general issue in learning control with artificial neural networks. The motor control problem is a continuous problem (in states and actions spaces) with dynamics and as such presents challenges which can be neglected in discrete-action problems without inertia such as the Atari games. In reinforcement learning, the learning agent has to figure out on its own (based on the rewards) the task's goal. In large continuous problems it can result in a time consuming exhaustive search. A common approach is to learn from demonstration, which shows the agent exactly what to do. But the construction of the demonstration database is hard and requires an expert, and the agent learns only what it sees, a fact that when transitioning to the on-line RL phase can confuse the exploration mechanism. The need to devise a method more smooth and which does not require the presence of an expert is evident. We proposed to merge the demonstrations in an on-line fashion with crude demonstrations which are easy to construct.

The other challenge that arises is the forgetting of old experience and its affect. One approach that can be taken to address that is to increase the size of the experience replay buffer, but it will always have a limit, and that also presents technical problems of handling such large chunks of memory.

We showed that the target network plays a key role in the forgetting phenomenon. By changing the update rate of the target network, the forgetting problem is prevented completely, without increasing the replay buffer capacity or introducing time expensive computations in order to keep important old samples.

The approach presented in this paper is goal-oriented, and will work for other goals beyond the static setting of striking a stationary puck. A natural step in this research direction is to apply our approach to a dynamic setting of a moving puck. We believe our approach is suitable for that task with no more than minor adaptations. Furthermore, we believe that our ideas can be used as the basis for learning end-to-end how to play a complete air hockey game.

References

[1] M. Abadi et al., "TensorFlow: A System for Large-Scale Machine Learning," in *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation*, vol. 16, 2106, pp. 265-283.

[2] P. Abbeel, and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proceedings of the Twenty-First International Conference on Machine learning*, ACM, 2004, p. 1.

[3] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and Autonomous Systems*, vol. 57, no. 5, 2009, pp. 469-483.

[4] C. G. Atkeson, and J. C. Santamaria, "A comparison of direct and model-based reinforcement learning," in *Proceedings of IEEE International Conference on Robotics and Automation*, vol. 4, IEEE, 1997, pp. 3557-3564.

[5] D. C. Bentivegna and C. G. Atkeson, "A framework for learning from observation using primitives," in *Robot Soccer World Cup*. Springer Berlin/Heidelberg, 2003, pp. 263-270.

[6] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang and W. Zaremba, "OpenAI gym," *arXiv preprint arXiv:1606.01540*, 2016.

[7] N. Chentanez, A. G. Barto and S. P. Singh, "Intrinsically motivated reinforcement learning," in *Advances in Neural Information Processing Systems*, 2005, pp. 1281-1288.

[8] M. P. Deisenroth, G. Neumann and J. Peters, "A survey on policy search for robotics," *Foundations and Trends® in Robotics*, vol. 2, no. 1–2, 2013, pp. 1-142.

[9] Y. Duan, X. Chen, R. Houthooft, J. Schulman and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *International Conference on Machine Learning*, 2016, pp. 1329-1338.

[10] R. M. French, "Catastrophic forgetting in connectionist networks," *Trends in Cognitive Sciences 3*, no. 4 (1999): 128-135.

[11] B. Goodrich and I. Arel, "Mitigating catastrophic forgetting in temporal difference learning with function approximation," in *2nd Multidisciplinary Conf. on Reinforcement Learning and Decision Making*, 2015.

[12] T. Hester et al., "Learning from Demonstrations for Real World Reinforcement Learning," *arXiv preprint arXiv:1704.03732*, 2017.

[13] J. Kirkpatrick et al., "Overcoming catastrophic forgetting in neural networks." in *Proceedings of the National Academy of Sciences*, vol. 114, 2017, pp. 3521-3526.

[14] J. Kober, J. A. Bagnell and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, 2013, pp. 1238-1274.

[15] S. Levine and P. Abbeel, "Learning neural network policies with guided policy search under unknown dynamics," in *Proceedings of Advances in Neural Information Processing Systems*, 2014, pp. 1071-1079.

[16] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[17] M. J. Mataric, "Reward functions for accelerated learning," in *Proceedings of The Eleventh International Conference on Machine Learning*, 1994, pp. 181-189.

[18] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proceedings of The International Conference on Machine Learning*, 2016, pp. 1928-1937.

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

[20] V. Mnih et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, 2015, pp. 529-533.

[21] K. Muelling, J. Kober and J. Peters, "Learning table tennis with a mixture of motor primitives." in *Proceedings of the 10th IEEE-RAS International Conference on Humanoid Robots*, 2010, pp. 411-416.

[22] A, Nair et al, "Massively parallel methods for deep reinforcement learning," *arXiv preprint arXiv:1507.04296*, 2015.

[23] A. Namiki, S. Matsushita, T. Ozeki and K. Nonami, "Hierarchical processing architecture for an air-hockey robot system," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2013, pp. 1187-1192..

[24] A. Y. Ng, D. Harada and S. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *Proceedings of the International Conference on Machine Learning*, vol. 99, 1999, pp. 278-287

[25] C, B, Partridge and M. W. Spong, "Control of planar rigid body sliding with impacts and friction," *The International Journal of Robotics Research*, vol. 19, no. 4, 2000, pp. 336-348.

[26] D. Pathak, P. Agrawal, A. A. Efros and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," *arXiv preprint arXiv:1705.05363*, 2017.

[27] S. Schaal, "Learning from demonstration," in *Proceedings of the Advances in Neural Information Processing Systems*, 1997, pp. 1040-1046.

[28] T. Schaul, J. Quan, I. Antonoglou and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[29] J. Schulman, P. Moritz, S. Levine, M. Jordan and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation." *arXiv preprint arXiv:1506.02438*, 2015.

[30] D. Silver et al., "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, 2016, pp. 484-489.

[31] D. Silver G. Lever, N. Heess, T. Degris, D. Wierstra and M. Riedmiller, "Deterministic policy gradient algorithms," In *Proceedings of the 31st International Conference on Machine Learning*, 2014, pp. 387-395.

[32] R. S. Sutton and A. G. Barto, "Reinforcement Learning: An Introduction," Cambridge: MIT press, 1998.

[33] Tesauro, Gerald, "Temporal difference learning and TD-Gammon," *Communications of the ACM*, vol. 38, no. 3, 1995, pp. 58-68.

[34] S. B. Thrun, "The role of exploration in learning control," *Handbook of Intelligent Control: Neural, Fuzzy and Adaptive Approaches*, 1992, pp. 1-27.

[35] S. B. Thrun, "Efficient exploration in reinforcement learning," Pittsburgh, PA, USA, Tech. Rep., 1992.

[36] G. E. Uhlenbeck and L. S. Ornstein. "On the theory of the Brownian motion," *Physical review*, vol. 36, no. 5, 1930, p. 823.

[37] H. Van, H., A. Guez and D. Silver, "Deep Reinforcement Learning with Double Q-Learning," in *AAAI*, 2016, pp. 2094-2100.

[38] C. J. Watkins and P. Dayan. "Q-learning," *Machine learning*, vol. 8, no. 3-4, 1992, pp. 279-292.

[39] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3-4, 1992, pp. 229-256.