Learning Control for Air Hockey Striking using Deep Reinforcement Learning

Ayal Taitler, Nahum Shimkin

Faculty of Electrical Engineering
Technion - Israel Institute of Technology

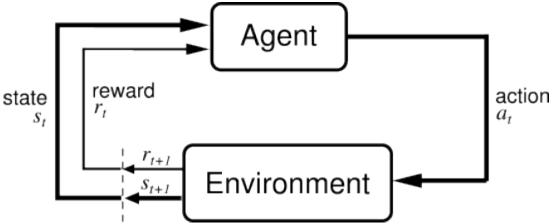
May 8, 2017

Overview

The Reinforcement Learning Setup

At each time step t:

1. Agent and environment in state s_t
2. The agent executes action a_t in the environment
3. The environment transitions to state s_{t+1} according to a_t and s_t
4. The agent observes state s_{t+1} and receives reward r_t
5. $t = t + 1$ and set $s_t = s_{t+1}$



RL Components

The RL goal is to maximize the expected accumulative reward:

RL Objective

$$\max_{\pi} \mathbb{E}^{\pi}\left[\sum_{t=0}^{T} \gamma^t r_t\right], \quad \pi : \mathcal{S} \to \mathcal{A}$$

The model is defined for the tuple $<\mathcal{S}, \mathcal{A}, \mathcal{P}, r>$, where

1. $\mathcal{A}$ - Action space
2. $\mathcal{S}$ - State space; initial state s_0
3. $\mathcal{P}$ - Transition model; $P(s'|s,a)$
4. r - Reward ; $r_t = r(s_t, a_t)$
5. γ - Discount factor

Q-Learning, Watkins (1989)

- Q-Learning - a learning version of the value iteration algorithm.
- At each step s_t, choose the action a_t which maximizes the function $Q(s_t, a_t)$.
  - Q is the estimated state-action value function it tells us how good an action is given a certain state.
- Formally: $Q(s_t, a_t) = r(s_t, a_t) + \gamma \max_a Q(s_{t+1}, a)$

Q-Learning, Watkins (1989)

- Q-Learning - a learning version of the value iteration algorithm.
- At each step s_t, choose the action a_t which maximizes the function $Q(s_t, a_t)$.
  - Q is the estimated state-action value function it tells us how good an action is given a certain state.
- Formally: $Q(s_t, a_t) = r(s_t, a_t) + \gamma \max_a Q(s_{t+1}, a)$
- The state-action value function can also be learned from off-policy samples $< s_t, a_t, r_t, s_{t+1} >$

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{old\ \ value} + \underbrace{\alpha}_{learning\ \ rate} \Big[\underbrace{r_t + \gamma \max_a Q(s_{t+1}, a)}_{target} - \underbrace{Q(s_t, a_t)}_{old\ \ value} \Big]$$

- Selected action: $\pi(s) = \operatorname{argmax}_a Q(s, a) + E$
  - E – Exploration

Deep Q-Learning

- State-action transitions $< s, a, r, s' >$ are sampled from
  - Full episodes roll-outs
  - Distribution function over an external memory

$$r = r(s, a), \quad s' \sim P(\cdot|s, a)$$

Deep Q-Learning

- State-action transitions $< s, a, r, s' >$ are sampled from
  - Full episodes roll-outs
  - Distribution function over an external memory

$$r = r(s, a), \quad s' \sim P(\cdot|s, a)$$

- The state-action value function can be approximated by a neural network (parametric function approximator)

$$Q^{\pi}(s, a) \approx Q(s, a; \theta)$$

Deep Q-Learning

- State-action transitions $< s, a, r, s' >$ are sampled from
  - Full episodes roll-outs
  - Distribution function over an external memory

$$r = r(s,a), \quad s' \sim P(\cdot|s,a)$$

- The state-action value function can be approximated by a neural network (parametric function approximator)

$$Q^{\pi}(s,a) \approx Q(s,a;\theta)$$

- Define the objective function to be the MSE of the Temporal Difference error

$$\mathcal{L}(\theta) = \mathbb{E}_{s,a,r,s' \sim P}\left[\left(\underbrace{r + \gamma \max_{a'} Q(s',a';\theta)}_{target} - \underbrace{Q(s,a;\theta)}_{prediction}\right)^2\right]$$

DQN & Double DQN Learning Rule, Van Hasselt et al. (2016)

- In order to reduce oscillation and increase stability, we use
  - Experience replay buffer $\mathcal{D}$
  - Additional freezed target Q-network $Q(s, a; \hat{\theta}^{-})$

$$\mathcal{L}^{DQN}(\theta) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}}\left[\left(r + \gamma \max_{a'} Q(s', a'; \hat{\theta}^{-}) - Q(s, a; \theta)\right)^2\right]$$

- Periodically update parameters $\hat{\theta}^{-} \leftarrow \theta$

DQN & Double DQN Learning Rule, Van Hasselt et al. (2016)

- In order to reduce oscillation and increase stability, we use
  - Experience replay buffer $\mathcal{D}$
  - Additional freezed target Q-network $Q(s,a;\hat{\theta}^{-})$

$$\mathcal{L}^{DQN}(\theta) = \mathbb{E}_{s,a,r,s'\sim\mathcal{D}}\left[\left(r + \gamma \max_{a'} Q(s',a';\hat{\theta}^{-}) - Q(s,a;\theta)\right)^2\right]$$

- Periodically update parameters $\hat{\theta}^{-} \leftarrow \theta$

- To reduce value overestimation we decouple the action selection from the action evaluation yielding

$$\mathcal{L}^{DDQN}(\theta) = \mathbb{E}_{\ s,a,r,s'\sim\mathcal{D}}\left[\left(r + \gamma Q(s',a';\hat{\theta}^{-}) - Q(s,a;\theta)\right)^2\right]$$

$$a' = \operatorname*{argmax}_{a} Q(s',a;\theta)$$

Deep Q-Network Learning Scheme

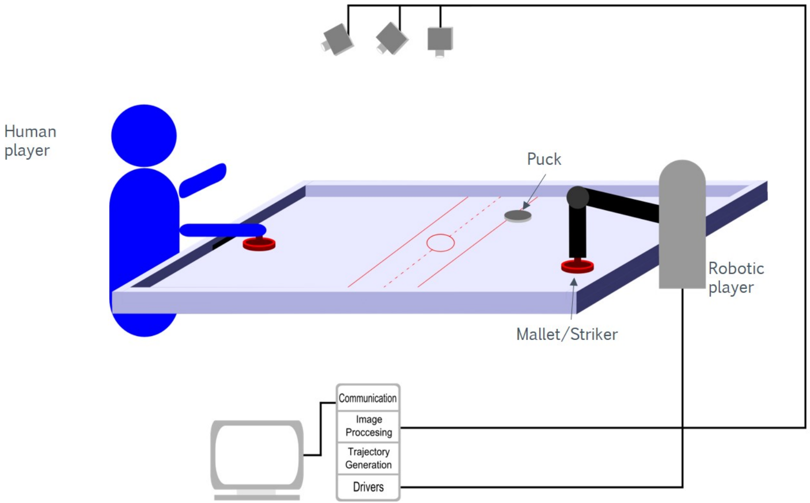
Initialize:

1. Experience replay buffer $\mathcal{D}$
2. Online network with weights θ
3. Target network with weights $\hat{\theta}^{-} \leftarrow \theta$

Start at state s_0 and repeat for M steps

1. In state s_t execute action $a_t = \max_a Q(s_t, a; \theta)$ / explore
2. Observe reward r_t and new state s_{t+1}
3. Store transition $< s_t, a_t, r_t, s_{t+1} >$ in experience replay buffer $\mathcal{D}$
4. Sample uniformly N transitions from $\mathcal{D}$
5. Update Q-network $Q(s, a; \theta)$ according to $\mathcal{L}(\theta)$
6. Set $s_t \leftarrow s_{t+1}$
7. update target Q-network with $\hat{\theta}^{-} \leftarrow \theta$ every C steps

The Air Hockey Physical Setup



Air Hockey Striking Problem Formulation

Goal

Finding a policy for the agent to strike the puck (skill), such that the puck will move in a desired attack pattern

Air Hockey Striking Problem Formulation

Goal

Finding a policy for the agent to strike the puck (skill), such that the puck will move in a desired attack pattern

Assumptions:

- Puck is stationary
- The Agent has known physical limitations
- The agent's physical model and puck-mallet collision model are unknown

Air Hockey Striking Problem Formulation

Goal

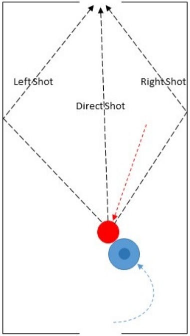
Finding a policy for the agent to strike the puck (skill), such that the puck will move in a desired attack pattern

Assumptions:

- Puck is stationary
- The Agent has known physical limitations
- The agent's physical model and puck-mallet collision model are unknown

Requirements:

- Maximum velocity after impact (puck)
- Puck is aimed to the center of the goal
- Puck's trajectory according to the selected skill



Agent's Dynamics

State space:

- Combination of the agent's (m) and puck's (p) positions and velocities

$$s_t = \left[m_x,\ m_{V_x},\ m_y,\ m_{V_y},\ p_x,\ p_{V_x},\ p_y,\ p_{V_y}\right]^T$$

Agent's Dynamics

State space:

- Combination of the agent's (m) and puck's (p) positions and velocities

$$s_t = \left[m_x,\ m_{V_x},\ m_y,\ m_{V_y},\ p_x,\ p_{V_x},\ p_y,\ p_{V_y} \right]^T$$

Action Space:

- Acceleration commands (motor torques) $\left[a_x,\ a_y \right]^T, \quad |a_{x,y}| \leq A_{max}$
- Discretized in order to fit the Q-networks scheme
- Boundary and zero actions are taken (among other) to include the known Bang-Zero-Bang profile

$$A = \left[-A_{max},\ -A_{max}/2,\ 0,\ A_{max}/2,\ A_{max} \right]^2$$

Agent's Dynamics

State space:

- Combination of the agent's (m) and puck's (p) positions and velocities

$$s_t = \left[m_x,\ m_{Vx},\ m_y,\ m_{Vy},\ p_x,\ p_{Vx},\ p_y,\ p_{Vy}\right]^T$$

Action Space:

- Acceleration commands (motor torques) $\left[a_x,\ a_y\right]^T, \quad |a_{x,y}| \leq A_{max}$
- Discretized in order to fit the Q-networks scheme
- Boundary and zero actions are taken (among other) to include the known Bang-Zero-Bang profile

$$A = \left[-A_{max},\ -A_{max}/2,\ 0,\ A_{max}/2,\ A_{max}\right]^2$$

Simulation dynamical model:

- 2-D 2^{nd} order kinematics (integrator) with constraints
- Collision models are ideal, with restitution coefficient $e = 0.99$

Time Optimal Control Formulation

Control problem

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

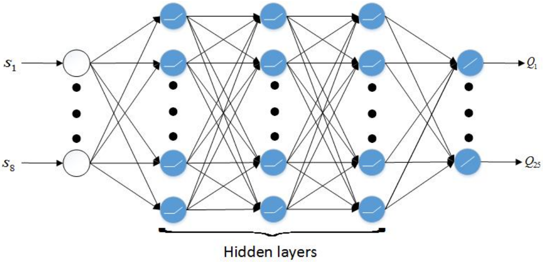
- $\phi(s_T)$ is the constraint on the final condition
- $f(s_k, a_k)$ is the physical model
- $S_{min/max}$, $A_{min/max}$ are constraints on the space and actions respectively
- s_0, s_T are the initial and final conditions
- The collision model is embedded within the state and final condition

The NN Controller

- **Input** - physical state vector of the game, i.e position and velocities

$$s_t \in \mathbb{R}^8$$

- **Output** - 25 Q values (5 actions in each axis)
- **Network** - Feedforward Neural Network with 4 Layers



- **Activations** - ReLU

Reward Definition

Reward Function

$$\begin{cases} r_t = -r_{time} & \text{if } s_t \text{ is not terminal} \\ r_t = r_c + r_v + r_d & \text{if } s_t \text{ is terminal} \end{cases}$$

Reward Definition

Reward Function

$$\begin{cases} r_t = -r_{time} & \text{if } s_t \text{ is not terminal} \\ r_t = r_c + r_v + r_d & \text{if } s_t \text{ is terminal} \end{cases}$$

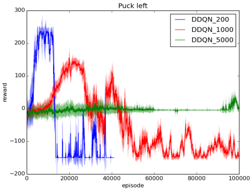
- r_c is a reward given for striking the puck
- r_v is a reward for maximal velocity in the direction of the goal

$$r_v = \text{sign}(V) \cdot V^2$$

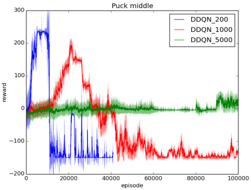
- r_d is a reward for the puck reaching the desired point at the opponent's goal

$$r_d = \begin{cases} c & |x - x_g| \leq w \\ c \cdot e^{-d(|x - x_g| - w)} & |x - x_g| > w \end{cases}$$

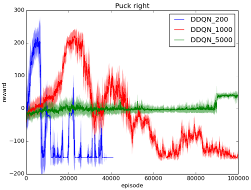
Applying the DQN scheme to the air hockey problem
(testing for different update periods)



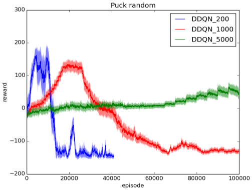
(a) Left



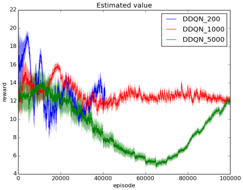
(b) Middle



(c) Right



(d) Random



(e) Estimated value

Challenges

Observations:

- DQN takes a long time to start to rise (no feature learning)
- Best policy learned is suboptimal
- Sharp drop in score values - obtaining oscillating policy
- Average value is noisy and oscillating
- Learning error (TD) diverges
- DQN is very inconsistent

Challenges

Observations:

- DQN takes a long time to start to rise (no feature learning)
- Best policy learned is suboptimal
- Sharp drop in score values - obtaining oscillating policy
- Average value is noisy and oscillating
- Learning error (TD) diverges
- DQN is very inconsistent

Possible explanations:

- Continuous state space
- Physical model dynamics
- Lack of rewards

Exploration/Exploitation in Continuous Domains

- ϵ-greedy

$$a_t = \begin{cases} a_t^* & \text{with probability } 1-\epsilon \\ \text{random action} & \text{with probability } \epsilon \end{cases}$$

- Local Exploration

$$a_t = a_t^* + \mathcal{N}_t$$

$\mathcal{N}_t$ - temporally correlated random process

Exploration/Exploitation in Continuous Domains

- ϵ-greedy

$$a_t = \begin{cases} a_t^* & \textit{with probability } 1-\epsilon \\ \textit{random action} & \textit{with probability } \epsilon \end{cases}$$

- Local Exploration

$$a_t = a_t^* + \mathcal{N}_t$$

$\mathcal{N}_t$ - temporally correlated random process

- ϵ-greedy gets filtered in physical systems with inertia, the system functions as a low pass filter
- Local exploration needs to be around the optimum to work

Learning Guidance with On-line Demonstrations

- Combine knowledge with present experience mechanism for one stage learning
  1. With probability ϵ_p instruct the agent to act according to $\pi(s)$ for a **full episode**
  2. Store transitions in replay buffer for learning

Learning Guidance with On-line Demonstrations

- Combine knowledge with present experience mechanism for one stage learning
  1. With probability ϵ_p instruct the agent to act according to $\pi(s)$ for a **full episode**
  2. Store transitions in replay buffer for learning

- Acting scheme:
  with probability ϵ_p perform a guided episode

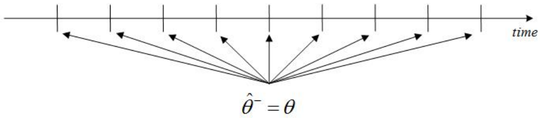
$$a_t = \pi_g(s_t)$$

with probability $1 - \epsilon_p$ act according to the E/E scheme

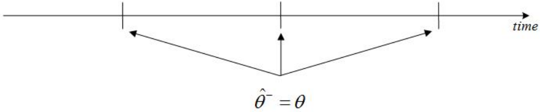
$$a_t = \begin{cases} \max_a Q(s_t, a) + \mathcal{N}_t & \textit{with probability } 1 - \epsilon \\ \textit{random action} & \textit{with probability } \epsilon \end{cases}$$

Fixed Target Update

- Samples in the replay buffer (dataset) are not stationary.
  - Fast updates can follow changes rapidly, but may diverge

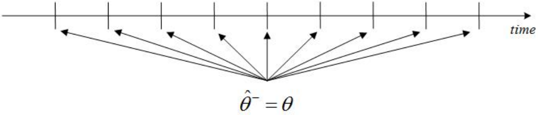


  - Slow updates can filter unstable changes, but may miss entire events

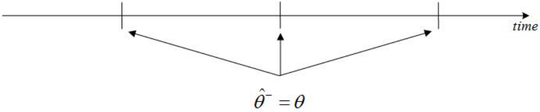


Fixed Target Update

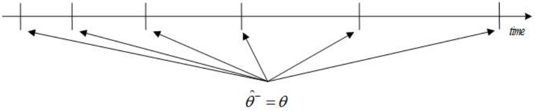
- Samples in the replay buffer (dataset) are not stationary.
  - Fast updates can follow changes rapidly, but may diverge



  - Slow updates can filter unstable changes, but may miss entire events



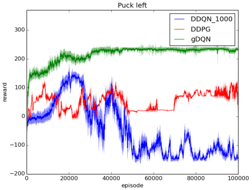
- Every update set: $C = C \cdot C_r, \quad C_r \geq 1$



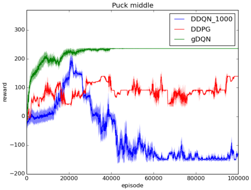
  - C – update rate

Guided-DQN

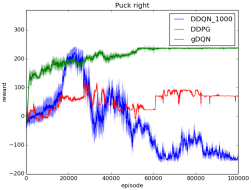
GDQN, Double DQN and Deep Deterministic Policy Gradients (DDPG)



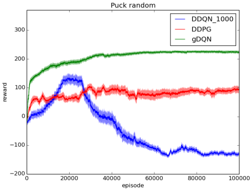
(a) Left



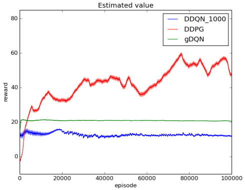
(b) Middle



(c) Right

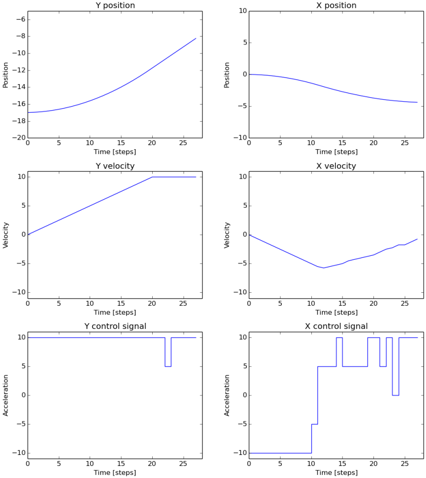


(d) Random

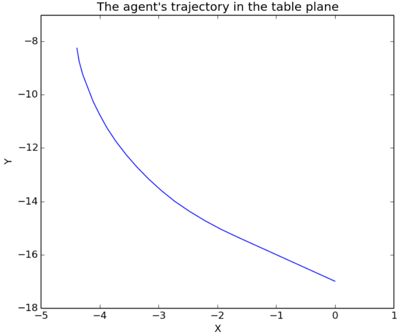


(e) Estimated value

Control Profile and Trajectory



(a) Profiles

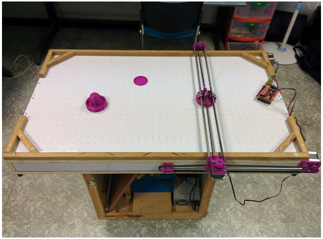


(b) Trajectory

OpenAI Striking Simulation

Summary

- Conclusion
  - Formalizing an optimal control problem as a learning problem.
  - Combining different methods of exploration.
  - Injecting prior knowledge into the exiting learning form experience framework.
  - Addressing the problem of non-stationarity in experience.
- Future Work
  - Extending the setting for a moving puck.
  - Extending the learning scheme for continuous actions.
  - Implementing the learning algorithm in physical environment.



Thats All Folks!

Thank You