Reinforcement Learning Case Study

Project on class [INF412] Autonomous Agents

Thomas Lagkalis

Electrical and Computer Engineering
Technical University of Crete

February, 2024

Table of Contents

Basic terms

- At time step t the agent is observing: $s_t \in S$
- Has to choose an action: $a \in A$
- Based on policy: $\pi : S \rightarrow A$
- Value function. A.k.a "how good" it is to be in a given state:

$$V_\pi(s) = \mathbb{E}[G|S_0 = s] = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_{t+1}|S_0 = s]$$

- Q function i.e. Q value is the expected discounted reward for executing action a at state s and following policy π thereafter.

$$Q^\pi(s, a) = \mathbb{E}[G|S_0 = s, a_0 = a] = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_{t+1}|S_0 = s, a_0 = a]$$

Q-Learning

It is proved that Q-learning converges to the optimum action-values with probability 1

Update rule:

$$Q_{s,a} = \begin{cases} Q(s_n, a)(1-\alpha) + \alpha(r_t + \gamma \max_a Q(s_{n+1}, a)) & \text{,if not a terminal state} \\ Q(s_n, a) & \text{,otherwise} \end{cases}$$

where:

- $a \in [0, 1]$ is the learning rate or step size. Determines to what extent newly acquired information overrides old information.
- Discount factor $\gamma \in [0, 1]$. Rewards in the distant future are weighted less

Q-Learning Algorithm

Algorithm 1 Q-Learning algorithm

Input

- number of episodes: numberOfEpisodes.
- initialized Q table Q(s,a) for all s,a.

Procedure

Get initial state s from the environment
for $i = 1, 2, 3, ..$ to numberOfEpisodes **do:**
while not a terminal state **do:**
sample action **a**.
make step in the environment based on action **a**
get the next sate s'
if s' is terminal state **do:**

$$Q(s, a) \leftarrow Q(s, a)$$

else do:

$$Q(s_n, a) \leftarrow Q(s_n, a)(1 - \alpha) + \alpha(r_t + \gamma \max_a Q(s_{n+1}, a))$$

end while
end for

Note: In the current implementation of the project the Q table is initialized with zeros.

Exploration vs Exploitation

The dilemma

we explore the environment by randomly sampling from the action space and thus letting the agent learn new features about the environment or we capitalize on knowledge already gained by being greedy?

Procedure of ϵ-greedy policy:

if current episode $<$ exploreEpisodes **do:**

return random a

else if random number $\in [0, 1] < \epsilon$ **do**:

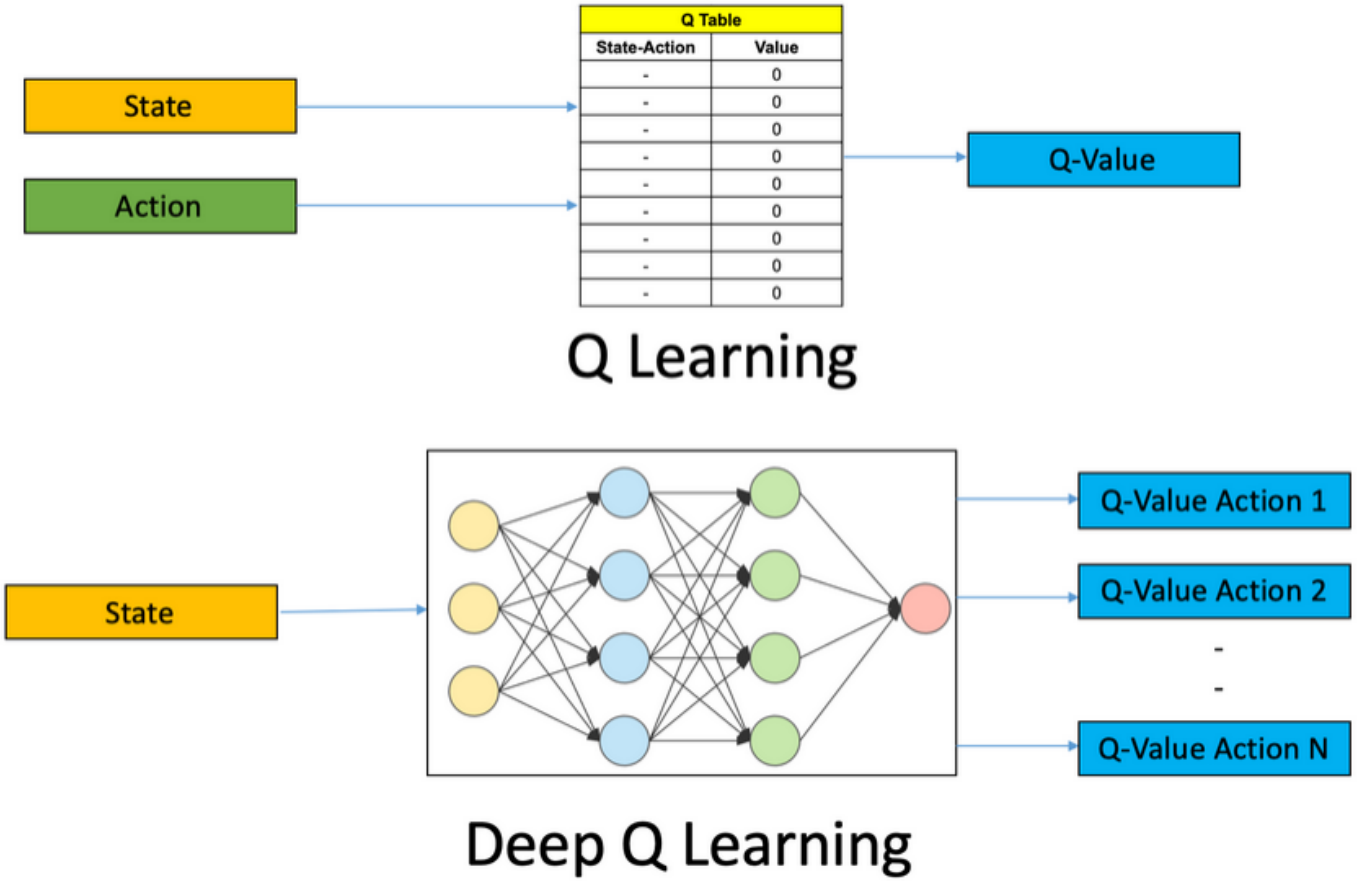
return random a

else:

return greedy action

DQN

In the Q-Learning algorithm all the Q-Values are stored in the memory.



Hence, we need a way to approximate the Q value (of each action) for each state for environments with large state-space.

DQN - Target Network

The problem

In order to train the NNs we need some ground truth values, but the predictions are changing over episodes.

Start with $Q_0(s, a)$ for all s, a.

Get initial state s

For k = 1, 2, ... till convergence

Sample action a, get next state s'

If s' is terminal:

$$\text{target} = R(s, a, s')$$

Sample new initial state s'

else:

$$\text{target} = R(s, a, s') + \gamma \max_{a'} Q_k(s', a')$$

Chasing a nonstationary target!

$$\theta_{k+1} \leftarrow \theta_k - \alpha \nabla_\theta \mathbb{E}_{s' \sim P(s'|s,a)} \left[(Q_\theta(s, a) - \text{target}(s'))^2 \right] \Big|_{\theta=\theta_k}$$

$$s \leftarrow s'$$

Updates are correlated within a trajectory!

DQN - Target Network

In order to address this problem we build two NNs, the online network which makes all the predictions and the target network which servers as the ground truth predictor. he target network is not updated in each step, but it's rather updated after a predefined number of steps (or episodes).

This technic is significantly improving the stability of the training process.

DQN - Experience replay

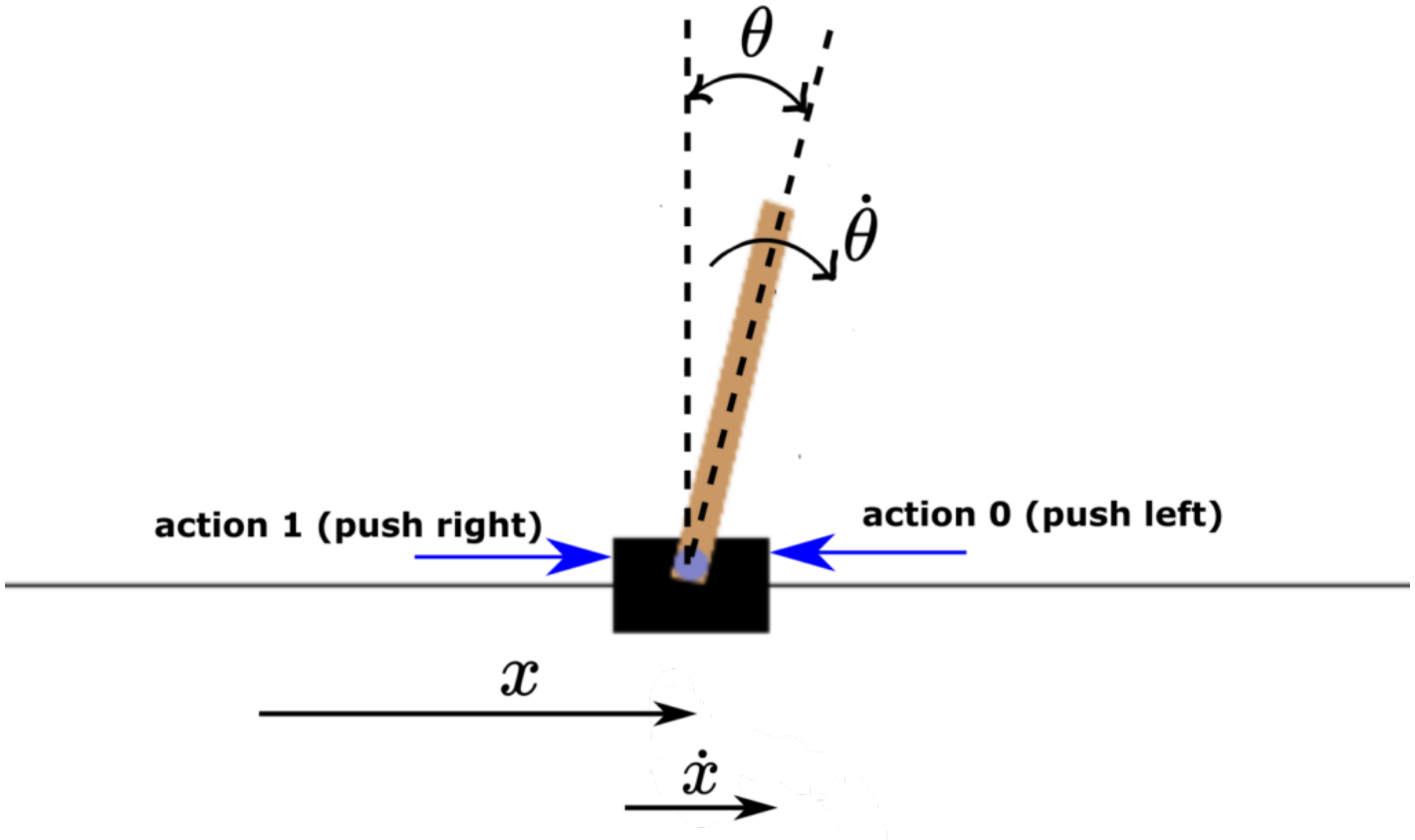
The Problem

Each experience (sate, action, reward, next state) depends on the previous experience acquired by the agent. The model may overfit to a subset of samples, and thus there might be a lack of generalization.

Solution: we store the agent's experiences at each time-step, in a data set pooled over many episodes into a replay memory. During each time step of the algorithm, we apply Q-learning updates, or minibatch updates, to samples of experience, drawn at random from the pool of stored samples (replay memory).

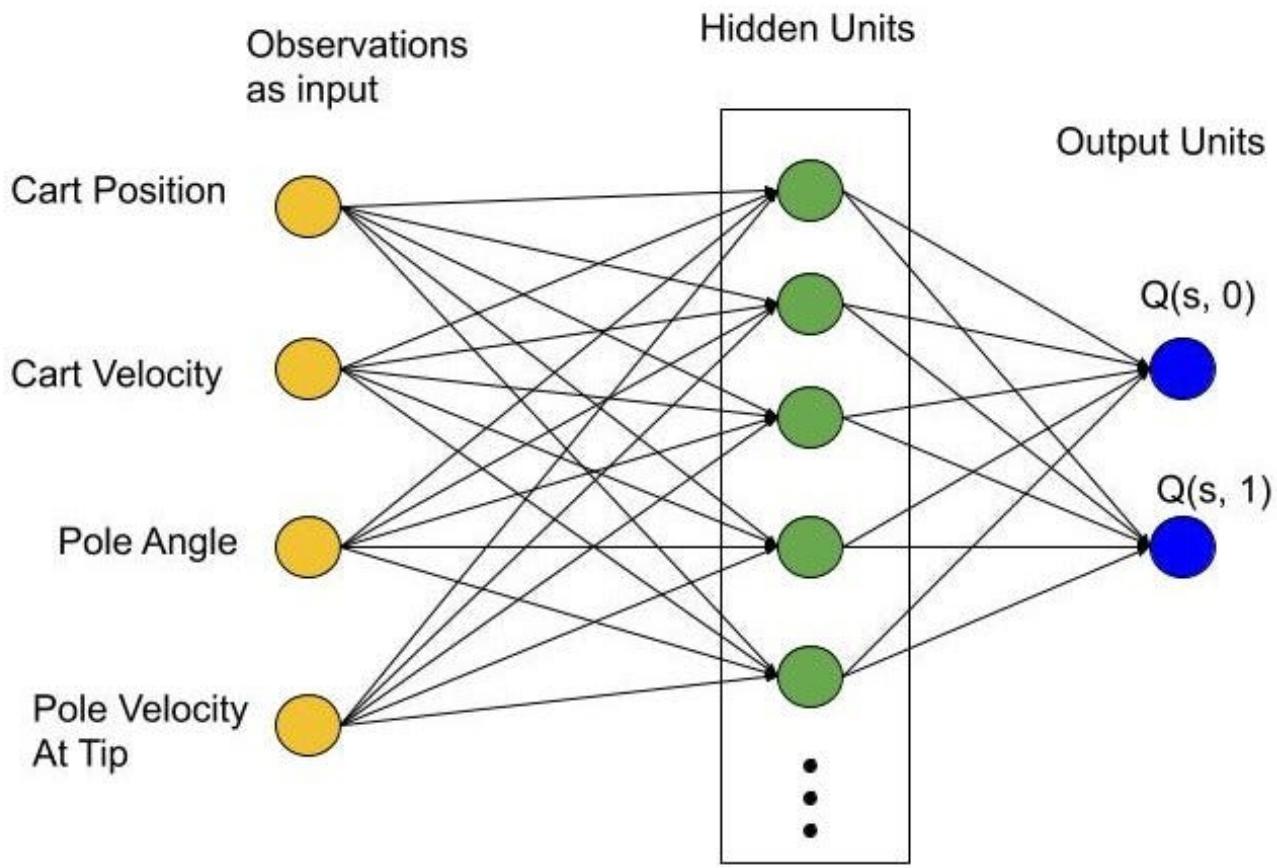
Cart Pole

- actions $= (0,1) =$ (push left, push right)
- *state* $= (x, \dot{x}, \theta, \dot{\theta})$



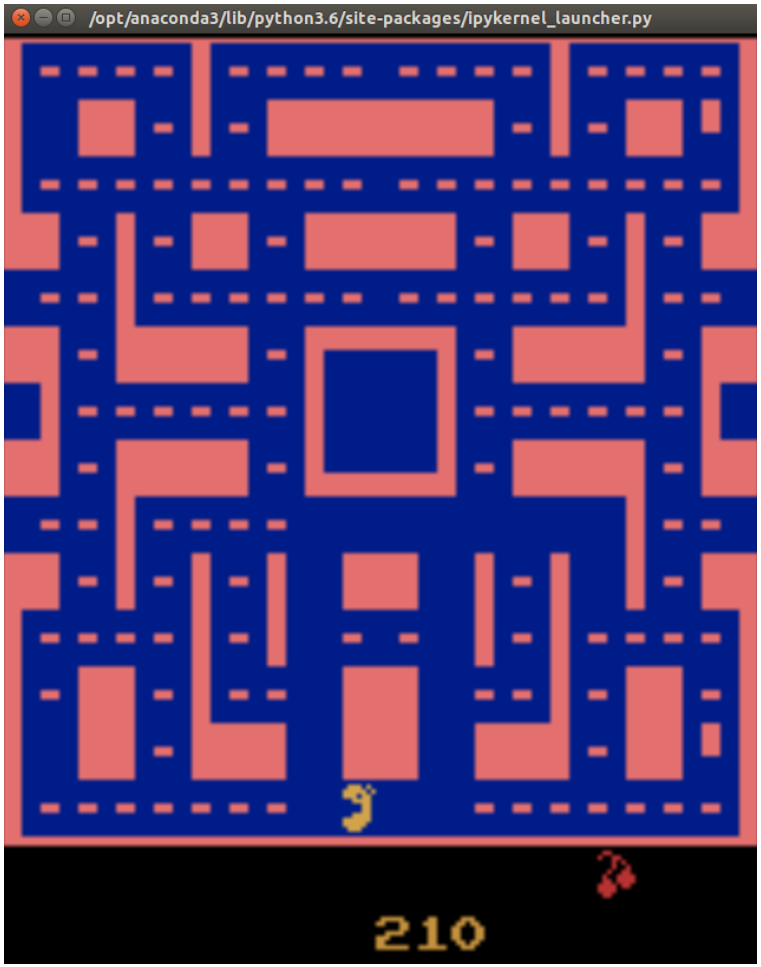
Cart Pole Model Architecture

Two hidden layers with 32 units and ReLu activation function.



Ms Pac Man

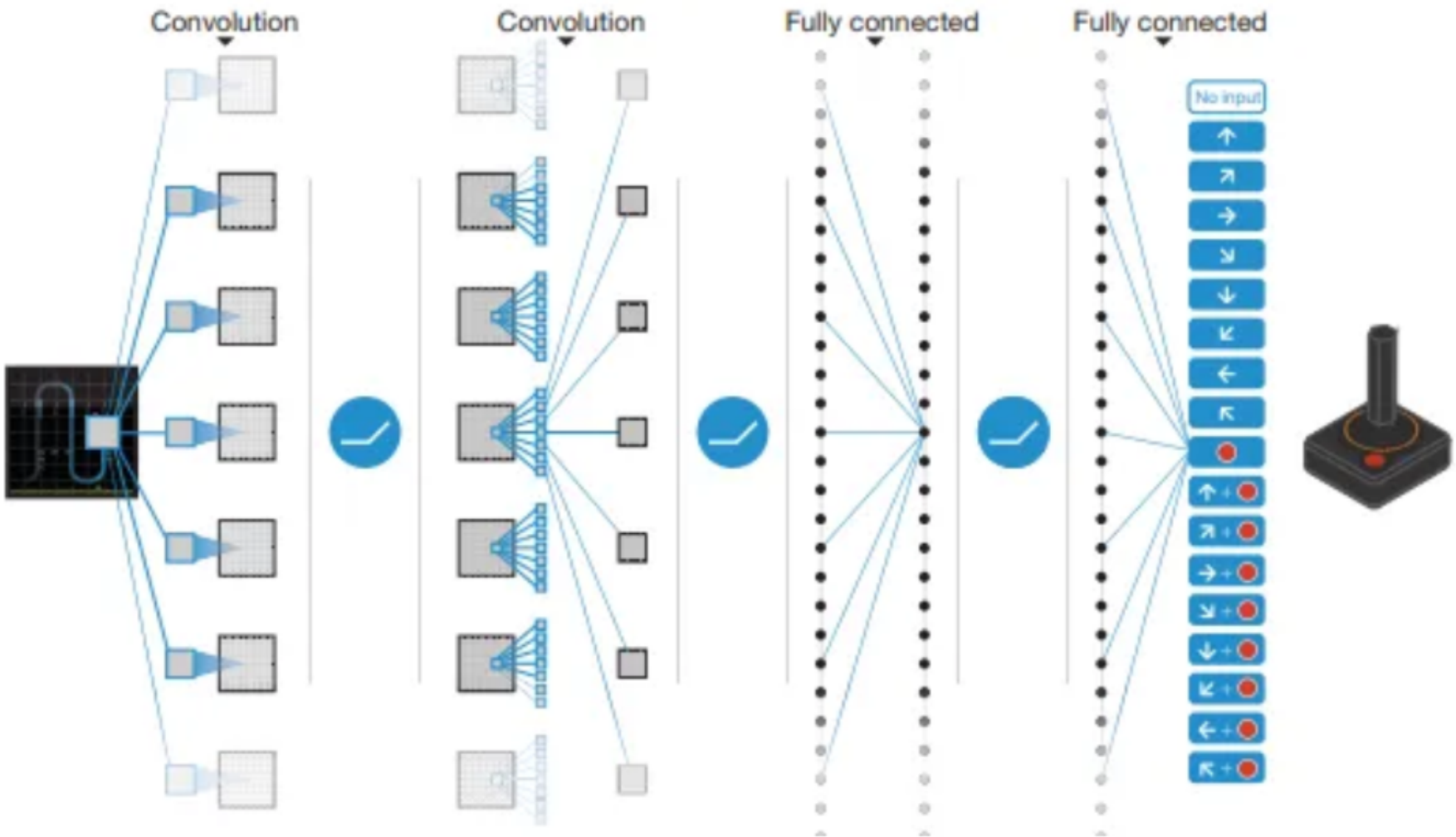
actions = (0, 1, 2, 3, 4, 5, 6, 7, 8) = (NOOP, UP, RIGHT, LEFT, DOWN, UPRIGHT, UPLEFT, DOWNRIGHT, DOWNLEFT)
state = RGB image, dimension 210x160x3



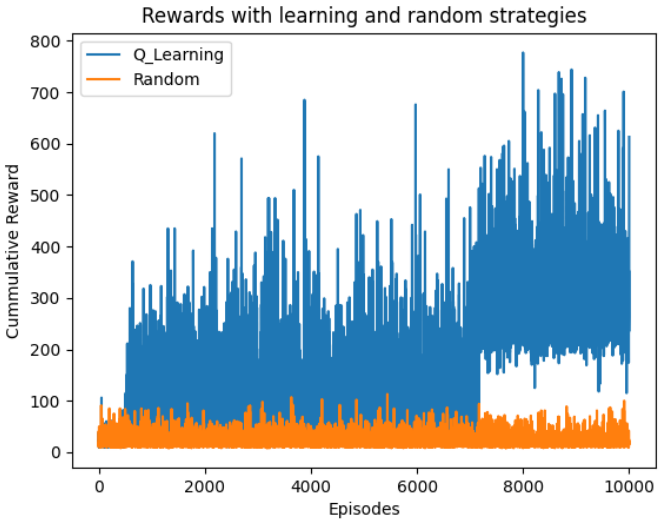
Ms Pac Man Model Architecture - 1

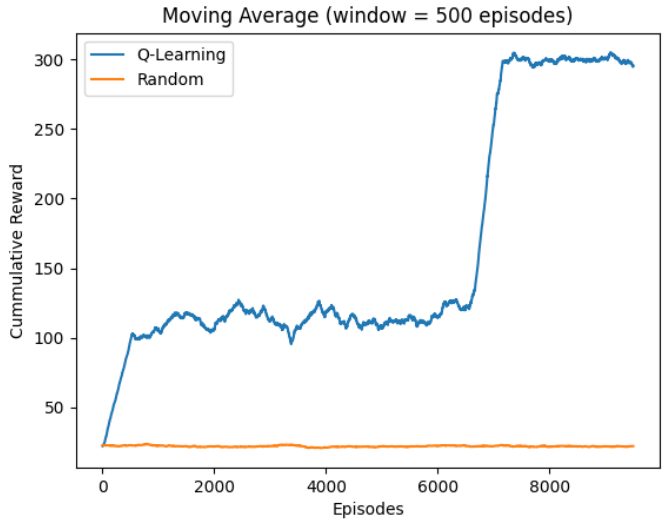
- Convolutional layer 1 consists of 32 8×8 filters
- Convolutional layer 2 consists of 64 4×4 filters
- Convolutional layer 3 consists of 64 3×3 filters
- 32 units Dense hidden layer
- 32 units Dense hidden layer
- Output layer

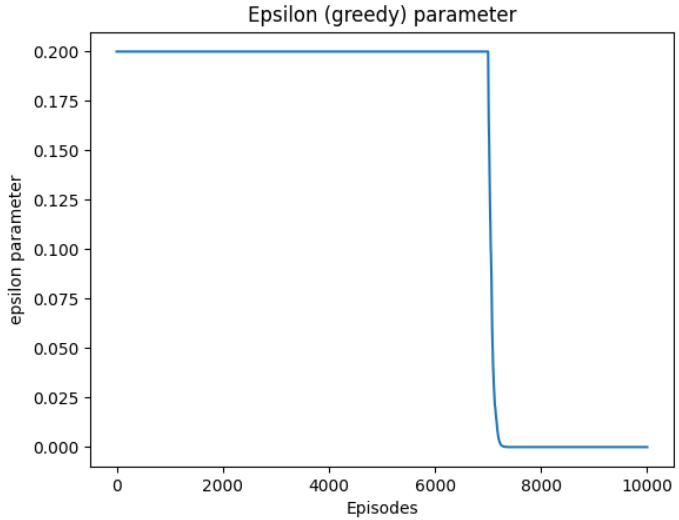
Ms Pac Man Model Architecture - 2



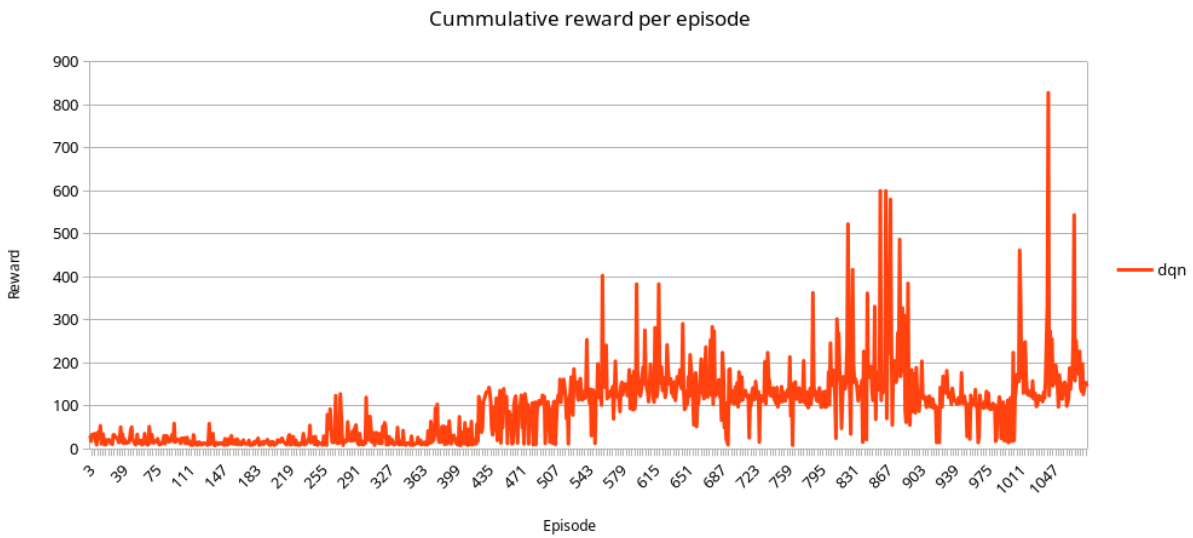
Car Pole with Q-Learning Results

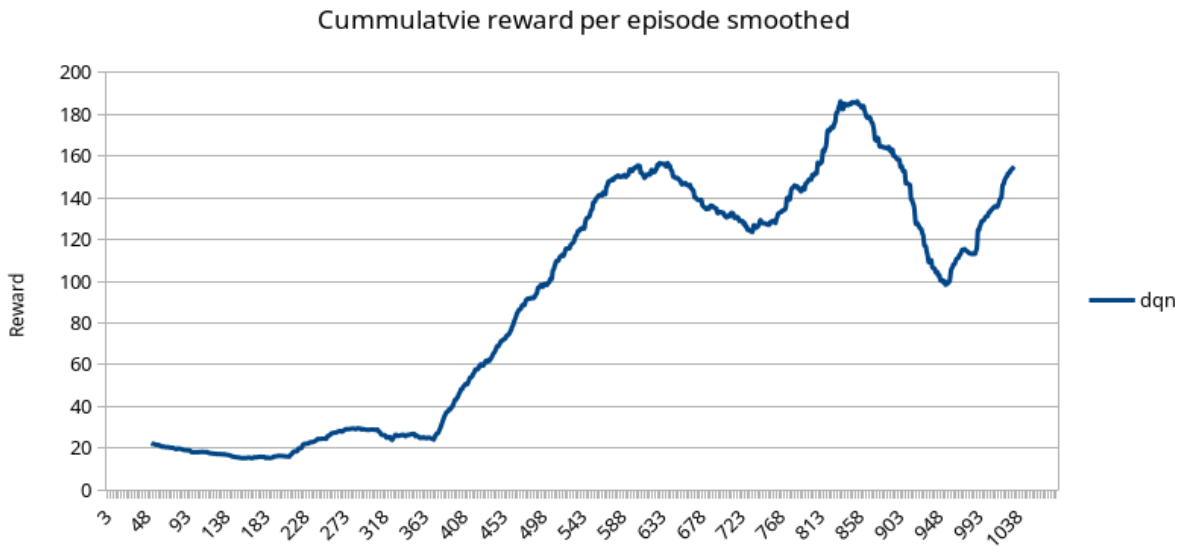




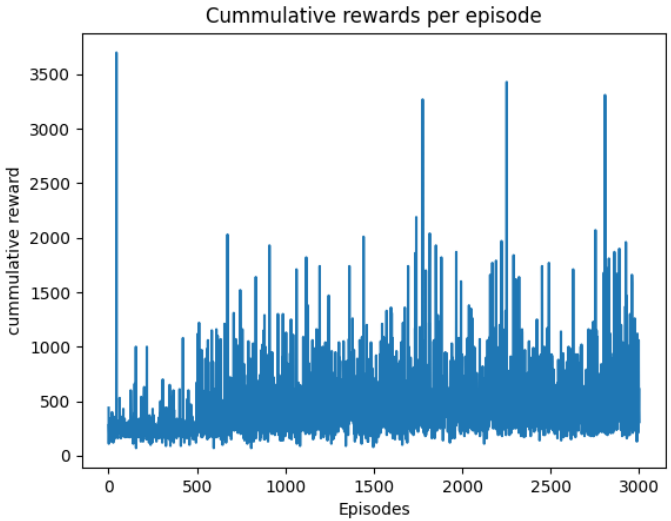


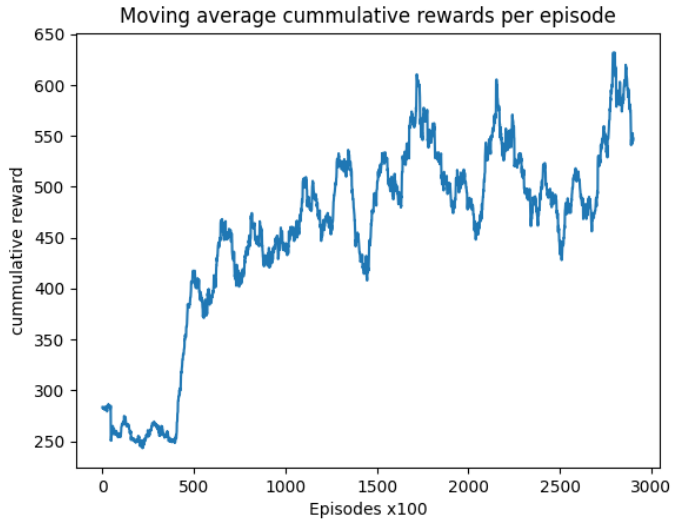
Cart Pole with DQN Results

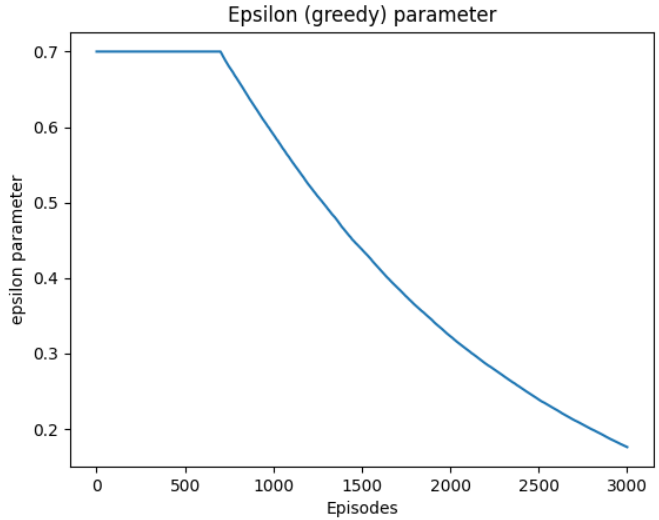




Ms Pac Man with DQN







Thank You!