UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR ROBOTICS
AND COGNITIVE SYSTEMS

Course: Reinforcement Learning (SS2020)
Prof. Dr. Elmar Rueckert & Honghu Xue, M.Sc.
Assignment Report Nr. IV
Date: 14. Juli 2021

Team Nr. 7
718651 Ricardo Sarau
737548 Fazli Faruk Okumus
737551 Mevlude Tigre
Contact: ricardo.sarau@student.uni-luebeck.de
fazli.okumus@student.uni-luebeck.de
mevluede.tigre@student.uni-luebeck.de

Task III: Going Deeper

1. **(i)** Below figure shows that training result of DDQN algorithm of each team-member. Training runs until test results converges 20 or more.

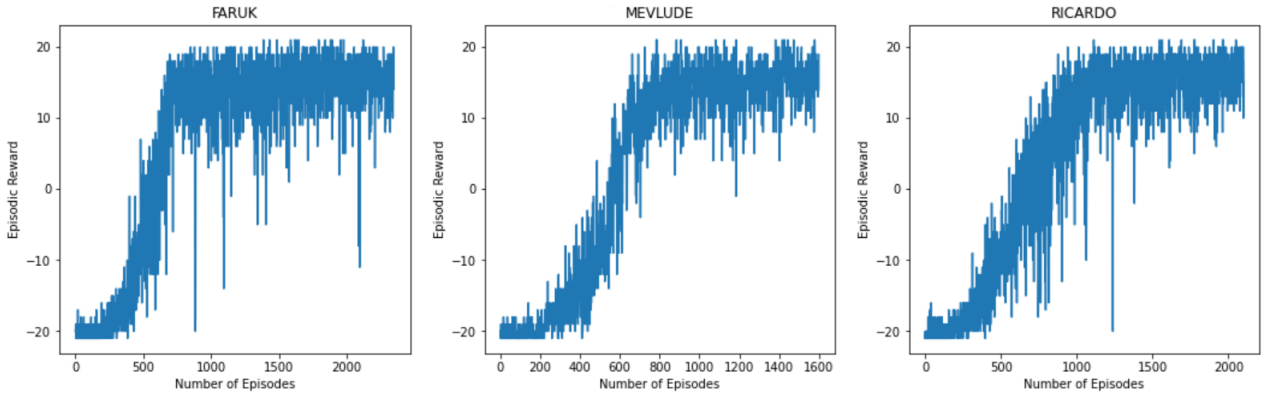

Figure 1 : Shown episodic reward of DDQN implementation on PongNoFrameskip-v4 environment

(ii)

a) The class **EpisodicLifeEnv** checks during training the current lives in the environment, compares them to the stored lives of the agent, updates them if necessary and can reset the environment. If one life is dropped, the episode is terminated. Normally a reset happens after all lives are lost and a true game over accrues, which then leads the class to fully reset the environment. If a reset happens after dropping from three to two lives, then the environment is reset back to the initial first state of the previous episode. The agent in this environment then starts with two lives instead of three, since the lives stored for the agent are copied from the current lives in the environment after a reset. This class designed to be call reset when reached terminal state, more specifically when the true game over.

b) Since we have **NoFrameskip** in our environment id, the immediate reward is processed in the class **MaxAndSkipEnv**. There are the rewards for corresponding frames are summed up to calculate the total reward. Total reward equal to summation of N different frames' rewards in our case N is equal to 4 by default. After the **MaxAndSkipEnv**, we pushing the reward to experience replay to use it later. Since the scale of scores varies greatly from game to game, we have to do something about the varying scales of rewards. Therefore we are are doing something called reward clipping, in which we clip the reward to be either $-1, 0$, or $+1$, depending on the sign of the actual reward received from the environment. This way, we limit the magnitude of the reward which can vary widely across the different environments. We can implement this simple reward clipping technique and apply it to our environments.So, all positive rewards are fixed to be 1 and all negative rewards to be -1, leaving 0 rewards unchanged. Clipping the rewards in this manner limits the scale of the error derivatives and makes it easier to use the same learning rate across multiple games. In the atari games, reward normalisation/scaling is performed so that games which use a moderate scoring system in single digits could be handled by the same neural network approximator. This reward clipping is only valid for the training phase.

c) When the environment is reset, the agent usually starts from the same initial state and therefore receives the same observation on reset. The agent may memorize or get used to the starting state in one game

level so much that they might start performing poorly they start in a slightly different position or game level. Sometimes, it was found to be helpful to randomize the initial state, such as sampling different initial states from which the agent starts the episode. To make that happen, **NoopRestEnv** is used that performs a random number of no-ops before sending out the first observation after the reset.

This class is designed to sample different initial states by performing random number of no-op actions on reset. No-ops are actions that can be done previous to the initial state under the assumption that they are action 0. Besides initiating environments with states the no-ops provided by this class are also used at the reset of environment. To bridge the time until the environment advertises done and the reset can be finished, no-ops are used as filler.

d) The class **MaxAndSkipEnv** returns only every N-th frame. What this means in practice is that only every fourth screenshot is considered, and then we form the "consecutive" frames that together become the input to the neural network function approximator. It allows us to speed up the training significantly by applying max to N observations (four by default) and return those as one observation for the step. This is because on intermediate frames, the chosen action is simply repeated and we can make an action decision every N steps. Alternatively one could process every frame with a Neural Network, but this would be quite a demanding operation and the difference between consequent frames is usually minor.

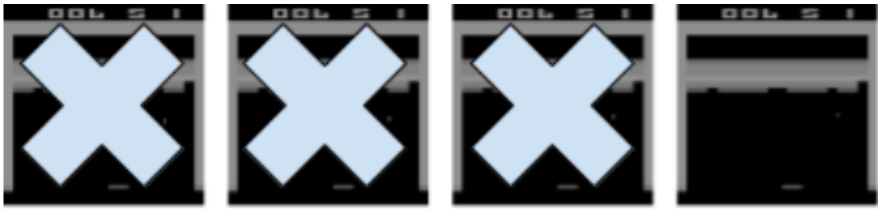

Figure 2 Only N-th frame is taking serious. In our case this is equal 4th by default

On the other hand, it takes the maximum of every pixel in the last two frames as shown in figure below and using it as an observation. Some Atari games have a flickering effect. For the human eye, such quick changes are not visible, but they can confuse a Neural Network.

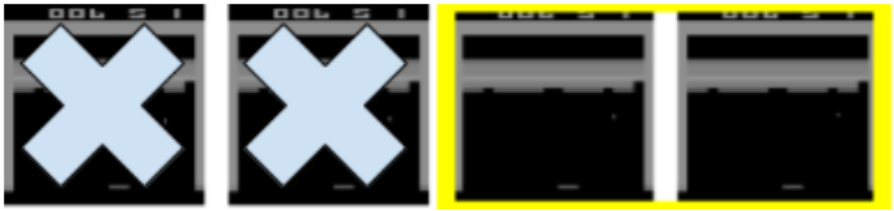

Figure 3 Taking the pixel-by-pixel (component-wise) maximum of the two images, which then get fed into as components

2. Below figure shows that training result of PER-DDQN algorithm. Training runs until test results converges 20 or more.

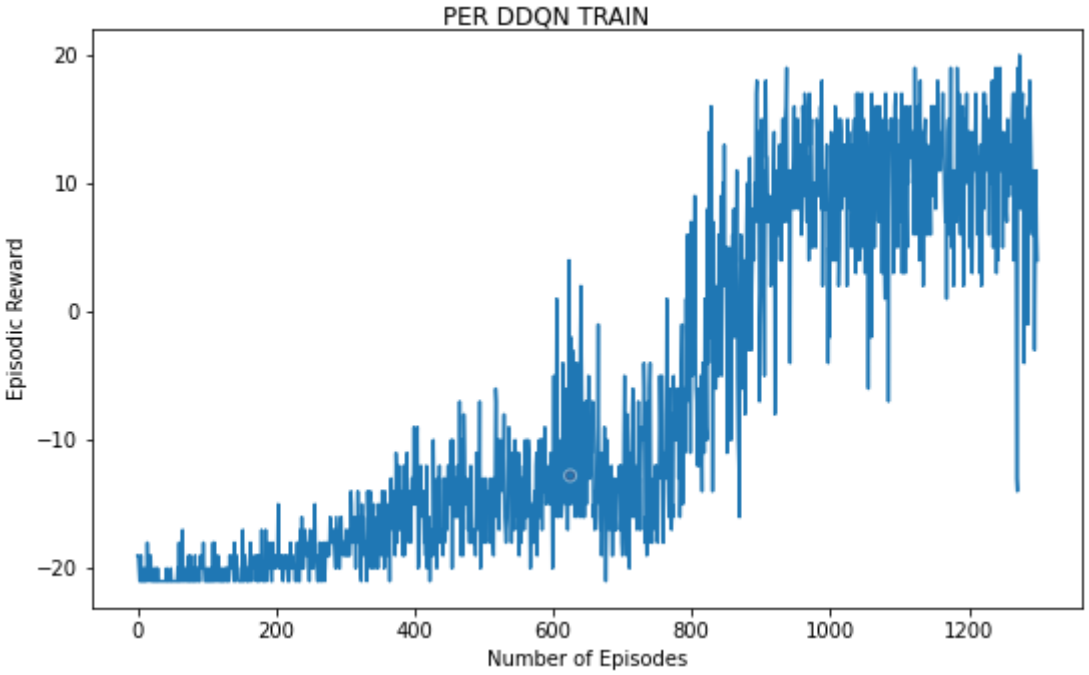

Figure 4 : Shown episodic reward of PER-DDQN implementation on PongNoFrameskip-v4 environment