For this project, you will work with the Tennis environment.

In this environment, two agents control rackets to bounce a ball over a net. If an agent hits the ball over the net, it receives a reward of +0.1. If an agent lets a ball hit the ground or hits the ball out of bounds, it receives a reward of -0.01. Thus, the goal of each agent is to keep the ball in play.

The observation space consists of 8 variables corresponding to the position and velocity of the ball and racket. Each agent receives its own, local observation. Two continuous actions are available, corresponding to movement toward (or away from) the net, and jumping.

The task is episodic, and in order to solve the environment, your agents must get an average score of +0.5 (over 100 consecutive episodes, after taking the maximum over both agents). Specifically,

- After each episode, we add up the rewards that each agent received (without discounting), to get a score for each agent. This yields 2 (potentially different) scores. We then take the maximum of these 2 scores.
- This yields a single score for each episode.

The environment is considered solved, when the average (over 100 episodes) of those scores is at least +0.5.

Background

To solve this problem I used the Deep Deterministic Policy Gradient (DDPG) algorithm with Prioritized Experience replay buffer.

Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient (DDPG) is an algorithm which concurrently learns a Q-function and a policy. It uses off-policy data and the Bellman equation to learn the Q-function, and uses the Q-function to learn the policy.

This approach is closely connected to Q-learning, and is motivated the same way: if you know the optimal action-value function $Q^*(s,a)$, then in any given state, the optimal action $a^*(s)$ can be found by solving:

$$a^*(s) = \ arg\ max_a Q^*(s,a)$$

When there are a finite number of discrete actions, the max poses no problem, because we can just compute the Q-values for each action separately and directly compare them. But when the action space is continuous, we can't exhaustively evaluate the space, and solving the optimization problem is highly non-trivial.

The way DDPG handles this, is by having two networks. One, called the actor, that approximates the best deterministic action, $a^*(s)$, for any given state. The second, called the critic, approximates the action-value function $Q^*(s,a)$.

Like DQN these networks change a lot and make learning instable, therefore local and target network with soft updates are also used.

So DQN network minimizes the following loss function:

$$L(\phi) = E[Q_\phi(s,a) - (r + \gamma(1-d)\max_a Q_\phi(s',a'))]$$

Where d indicates whether the state is terminal.

While DDQN critic network minimizes the following loss function:

$$L(\phi) = E[Q_\phi(s,a) - \left(r + \gamma(1-d)Q_{\phi target}\left(s', \mu_{\theta target}(s')\right)\right)]$$

Where $\mu_{\theta target}$ is the target actor network. The DDQN actor network maximizes the following function:

$$J = \frac{1}{n}\sum\nolimits_n Q_\phi(s,a)$$

Which is the same as minimizing:

$$J = -\frac{1}{n}\sum\nolimits_n Q_\phi(s,a)$$

Prioritized Experience replay buffer

Experience replay lets online reinforcement learning agents remember and reuse experiences from the past. However, this approach simply replays transitions at the same frequency that they were originally experienced, regardless of their significance.

In the presented environment were positive rewards are sparse and where a reward for an action has a delay, this represents a problem.

Prioritized experience replay buffer allows for the replay of more important samples more frequently and, therefore, learn more efficiently. The central component of prioritized replay is the criterion by which the importance of each sample is measured. The way I measured the importance was with the TD-error. For every new sample, the importance corresponds to:

$$R + Q_{\phi target}(s',a') - Q_\phi(s,a)$$

Where R represents the sample reward. At each learning step this value is updated.

However, if we use greedy priority then samples that have a low TD-error the first time, probably will never be visited again. And if the same samples are always visited then this can lead to overfit. A better approach would be to use stochastic sampling method:

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

where $p_i > 0$ is the priority of transition i. The exponent α determines how much prioritization is used, with α = 0 corresponding to the uniform case. Nevertheless, to prevent that low error sample not being visited we add a constant 0.01 to the calculated error.

Because DDPG is derived from an expectation over all experiences we need to account for the change in the distribution, updates to the network are weighted with importance sampling weights:

$$w_i = \left(\frac{1}{N} * \frac{1}{P(i)}\right)^{\beta}$$

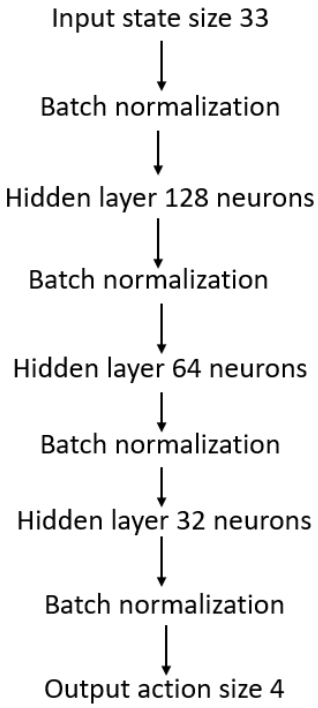
Otherwise the prioritized replay would introduce bias, because it changes the distribution in an uncontrolled fashion, and therefore changes the solution that the estimates will converge to.

Experiment

Neural Network structure

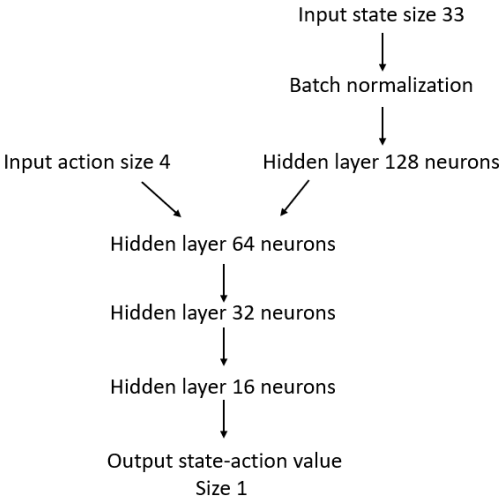
As described in the background section, DDPG has an actor network, which given an states outputs the best action and a critic network which given a state and an action evaluates how good the action was. To approximate these networks I used a Neural Network.

The actor network has the following structure:

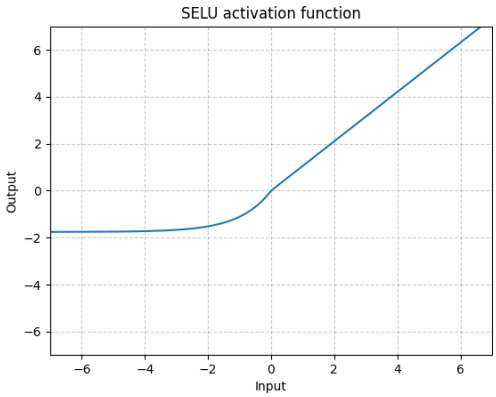


The activation functions used were selu for all layers except the last which used tanh function since the action space was between -1 and 1.

The critic network has the following structure:



The activation function use was selu for all layers. Even though selu is normally used for positive outputs, and we do have negative outputs of -0.01 when the ball is missed, the selu function looks like this:



As you can see, although it favors positive values, it allows for small negative values which corresponds to the environment reward system.

The number of neurons in the input layer represent the state space and the number of neurons for the output layer represent the action space. For this specific problems these number cannot be changed.

Hyperparameters

The following hyperparameters were used:

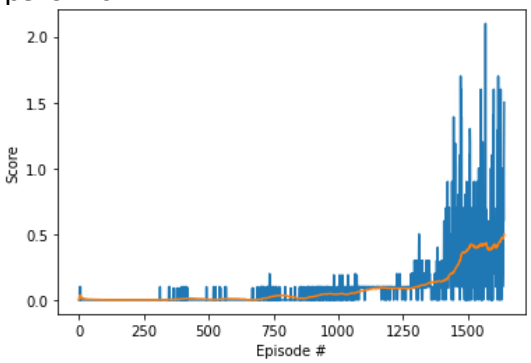
- Tau: 1e-3
- Gamma: 0.99
- Learning rate actor: 1e-4
- Learning rate critic: 1e-3
- Batch size: 128
- Replay buffer size: 1e5

- Learn every: 1, meaning that for 1 frames we collect the tuple (state, action, reward, next state, done) and then we perform one step of learning, were we calculate the gradients and update the networks.
- Alpha 0.3 or 0
- Beta 1

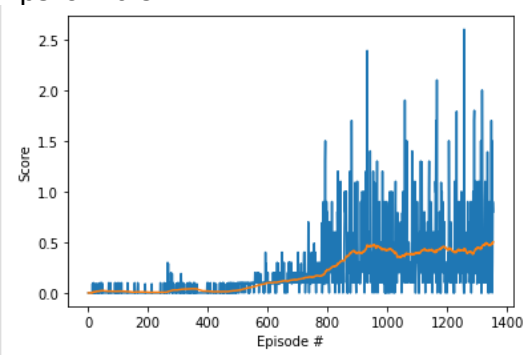
Results

On the following Figure we can see the obtained results. On the right, we have a model with no prioritized experience replay and on the left we have a model with prioritized experience replay with a epsilon of 0.3.

Epsilon=0



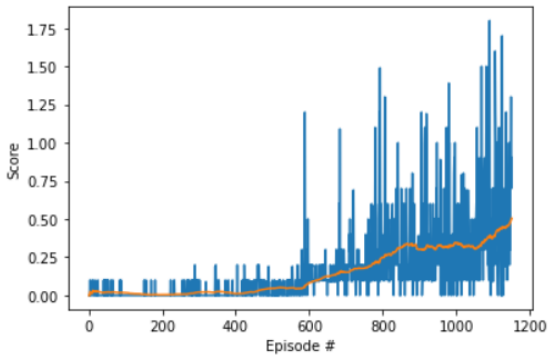
Epsilon=0.3



As we can see, the prioritized experience replay clearly increased the performance of the model, although it also made the model more instable, as we can see by looking at the variations of the average line. This variation might be because the model is trained more often with some samples than other which might lead to some overfit. This overfit is then corrected when the other samples increase their error and are then prioritized.

As we can see from the following Figure, if we lower the epsilon to 0.2 then the variation decreases and performance increases even further.

Epsilon=0.2



The trained actor and critic network can be found on the files ‘actor_local.pt’ and ‘critic_local.pt’, respectively.

Conclusion

As I showed, by fine tuning the networks and hyperparameters convergence can be achieved for this problem. And using prioritized replay may have a big impact in performance.

As future work I would like to implement a cooperative agent. In the current state an agent only cares about passing the ball over the net regardless of either the other agent can respond, meaning they are competitive. I think cooperativeness could be achieved by instead of using single rewards, using cumulative rewards. This way the agent would be rewarded for keeping the ball more time on game.

Afterwards, I would like to apply the AlphaZero algorithm to the agents, with the unified views.