Reinforcement Learning - Project assignment 3

For this assignment two reinforcement learning agents are trained to play tennis against each other. The goal for each agent is to play the ball over the net towards the other agent. The provided environment is based on a Unity Framework and is called tennis environment. In this setup **2 agents** are used, which are trained independently from each other. Each agent gets the environmental state for training from his own perspective. Each agent uses a **deep deterministic policy gradient** (DDPG) algorithm, which is implemented with **experience replay** and **double Q-learning** including an **actor** and **critic network**. The DDPG algorithm is used for a **continuous action space**. The goal for this setup is to reach a mean score of 0.5 points in average over 100 episodes.

1. Learning Algorithm

Reinforcement Learning is about to teach an agent to interact with an environment. The agent gets the current state of an environment and return the action with the goal to maximise a cumulative reward.
The chosen **multi agent** approach sets the agents up and orchestrates the actions and learning steps for each agent. It makes sure, that each agent gets only the information from the agents own perspective. So the agents learn and act independently from each other. This is a **competitive environment**, where each agent acts towards maximising its own reward.
The **deep deterministic policy gradient** (DDPG) algorithm extends the ideas of the Double-Q learning to the continuous action space (Lillicrap et al. 2015). DDPG is an actor-critic, model-free algorithm based on the deterministic policy. The DDPG algorithm uses an **actor-critic** architecture for function approximation to learn a policy and value function. As function approximation an artificial neural network is used each for the actor and the critic. The actor learns policies, while the critic learns to "criticise" the actor's current policy. The critic learns the state-value function for the current policy to evaluate the current action (Lillicrap et al. 2015, Sutton and Barto 2018). DDPG allows to explore the environment independently from the learning algorithm by adding noise to the actor policy. The **Ornstein-Uhlenbeck process** is used to generate this additional noise to generate temporally correlated exploration (Lillicrap et al. 2015).
Double Q-learning introduces a second neural network, which separates learning and acting. This approach reduces observed overestimations and has shown a better performance on several use cases (van Hasselt et al. 2015). Double Q-learning results in a more stable and reliable learning. With **experience replay** the agent experience (state) at each time step is stored in a replay buffer. This replay buffer aggregates state, action, reward tuples over many episodes of the simulation. A batch for the agent training is uniformly sampled from this replay buffer. So the agent learns from the action, state, reward tuple independently (Sutton and Barto 2018). This allows to learn the agent more efficient, since there is not much correlation between the experiences.

2. Model architecture and hyperparameters

The implementation of the neural network is quite similar for the actor and the critic network. The agent has for the actor and critic network a **local and target network**, which separates the update process. For each model two **fully connected neural network** are used with 2 hidden layers with 128 nodes (layer 1) and 128 nodes (layer 2). As activation functions **ReLu**s are used and for the actor model output tangent hyperbolic function (tanh) is used as activation function. Tangent hyperbolic function has the advantage that the gradient is steeper, so the values have a tendency to be +1 or -1. This can be an advantage, when the action needs a clear separation. As an optimizer the Adam optimizer is used with a learning rate of 0.001 for the actor network and 0.001 for the critic network. The **weight decay** determines the regularisation of the critic network optimisation and is set 0. So there is no additional parameter for the loss function used.

The reinforcement learning agent is trained with an **discount-factor** gamma of 0.99. For tau, an interpolation parameter for updating the double den, a value of 0.001 was used. The **replay buffer**

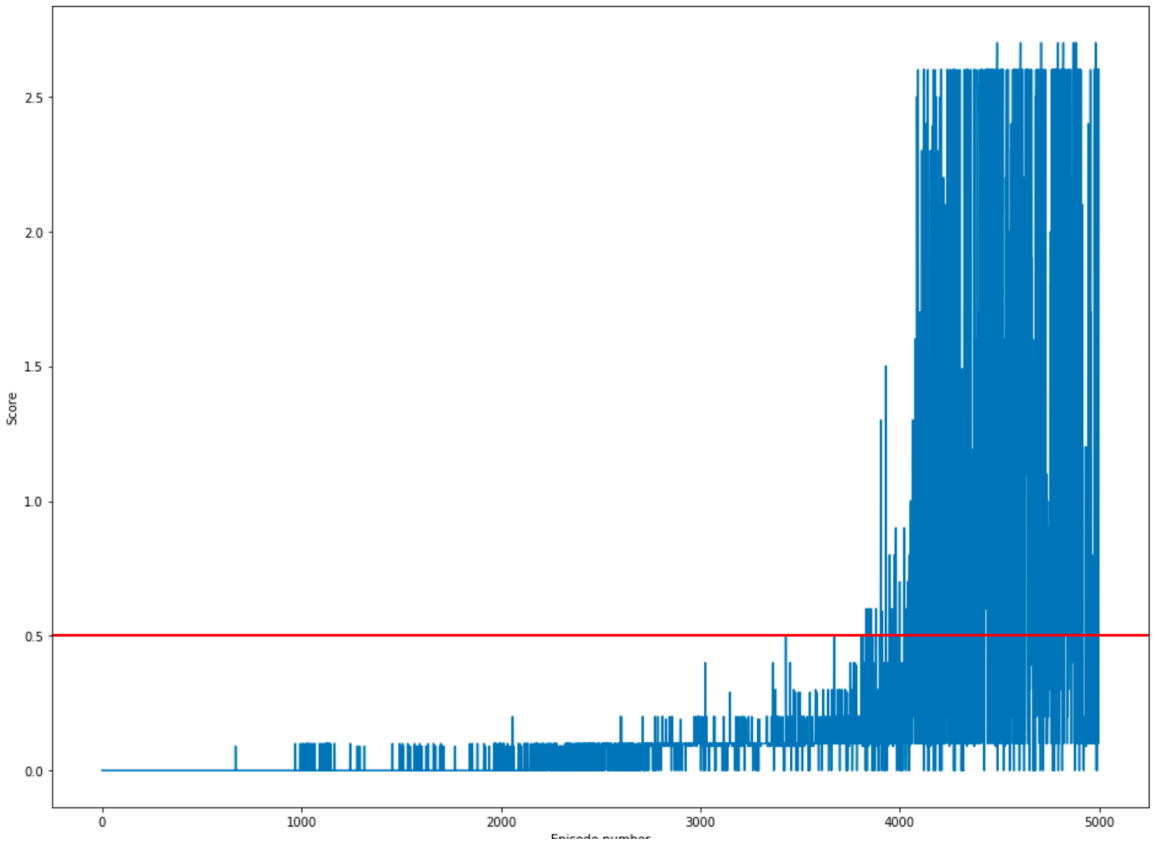
uses a buffer size of 10000 samples, where a batch of 128 samples are randomly picked for the training of the neural network.

Hyperparameter	Value
Buffer size	1E+05
Batch size	128
Gamma	0.99
Tau	1E-03
Learning rate (Actor)	1E-03
Learning rate (Critic)	1E-04
Weight Decay	0.0

3. Results

It needs with this setup 4133 **episodes** to achieve an average score **above 0.50 points**. From my experience with different setups, this takes so long, because sigma to add some noise was at 0.2. With a lower value or without adding noise at all, both agents learn much faster to play. In the beginning both agents take around 3000 episodes until they start to learn to play the game. Afterwards the agents need another 1000 episodes of small increment learning until they reach the score limit of 0.5. Around episode 4000 the agents learn to play, why the score got to the maximum possible of 2.7. Then most of the games got a high score with an peak average of 1.64 points at episode 4900.

Episode	Average score	Max score	Median score	Min score
500	**0.00**	0.00	0.00	0.00
1000	**0.00**	0.10	0.00	0.00
1500	**0.00**	0.10	0.00	0.00
2000	**0.01**	0.10	0.00	0.00
2500	**0.07**	0.10	0.09	0.00
3000	**0.10**	0.20	0.10	0.00
3500	**0.12**	0.50	0.10	0.01
4000	**0.26**	1.50	0.20	0.01
4133	**0.50**	2.60	0.25	0.00
4500	**1.27**	2.70	1.05	0.00
4900	**1.64**	2.70	1.90	0.00



4. Future ideas

Besides the implemented DDPG, there exist a plethora of algorithms, which can be implemented and tested for deep reinforcement learning for continuous control (Duan et al. 2016), as mentioned in the previous assignment. Especially the Distributed Distributional Deterministic Policy Gradients (DD4G, Barth-Maron et al. 2018) should be implemented to gain a more stable agent..
Also the current DDPG implementation can be improved by some implementation of the rainbow algorithm (Hessel et al. 2018). Possible improvements could be enabled by prioritised replay buffer or noisy networks.
In this implementation both agents learn independently to play tennis. An other approach could be to change from the competitive environment towards a cooperation environment, where only one neural network implementation is used and each agents learns from the states and actions of the other one.

5. Literature

Barth-Maron, Gabriel, et al. Distributed distributional deterministic policy gradients. arXiv preprint arXiv:1804.08617, 2018.

Duan, Yan, et al. Benchmarking deep reinforcement learning for continuous control. In: International Conference on Machine Learning. 2016. S. 1329-1338.

Hessel, Matteo, et al. Rainbow: Combining improvements in deep reinforcement learning. In: Thirty-Second AAAI Conference on Artificial Intelligence. 2018.

Lillicrap, Timothy P., et al. Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971, 2015.

Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. MIT press, 2018.

Van Hasselt, Hado, Arthur Guez, and David Silver. "Deep reinforcement learning with double q-l earning." Thirtieth AAAI conference on artificial intelligence. 2016.