Reinforcement Learning - Project assignment 2

For this assignment a reinforcement learning agent is trained to move a double-jointed robot arm to a target goal. The goal for the robot arm is to stay inside the target goal for as many time steps as possible. The provided environment is based on a Unity Framework and is called reacher environment. In this setup **20 double-joined robot arms** are used for the actor training. The agent uses a **deep deterministic policy gradient** (DDPG) algorithm, which is implemented with **experience replay** and **double Q-learning** for each, an **actor** and **critic network**. The DDPG algorithm is used for a **continuous action space**. The goal of each agent is to reach a mean score of 30 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 **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 3 hidden layers with 400 nodes (layer 1) , 300 nodes (layer 2) and 128 nodes (layer 3). For the Actor networks, a **batch normalisation** layer is used. 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 0.0001 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.

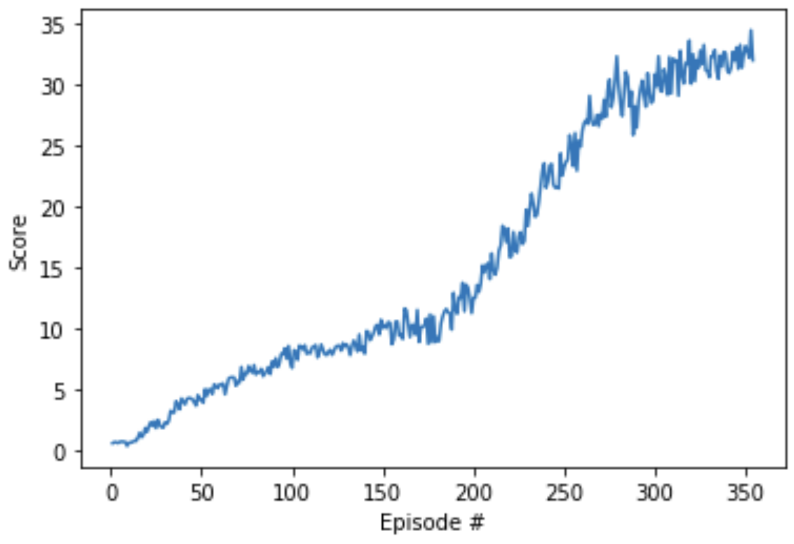
The reinforcement learning agent is trained with an **discount-factor** gamma of 0.99. For tau, an interpolation parameter for updating the double den, a value of 0.001 was used. The **replay buffer** uses a buffer size of 100000 samples, where a batch of 200 samples are randomly picked for the training of the neural network.The replay buffer is faster filled up by the parallel sampling of 20 robot arms. The Agent is trained with an **update frequency** of 20. So every 20th step, the agent learns from the sampled experiences from the replay buffer. The learning steps will be repeated 10 times defined by the **update steps**.

Hyperparameter	Value
Buffer size	1E+06
Batch size	200
Gamma	0.99
Tau	1E-03
Learning rate (Actor)	1E-03
Learning rate (Critic)	1E-04
Weight Decay	0.0
Update frequency	20
Update steps	10

3. Results

The agent needed **354 steps** to achieve an average score **above 30.00 points**. There were a lot of different implementations with bigger networks (2048, 2048, 1024 nodes), much bigger batch sizes (512, 1024) and different learning rates with weight decay. All those different implementations didn't reach the limit of 30.00 points. With the current implementation and hyperparameters, the agent is quite similar to the original DDPG agent (Lillicrap et al. 2015).

Episode	Average score	Max score	Median score	Min score
50	**2.33**	4.58	2.10	0.43
100	**4.29**	8.61	4.44	0.43
150	**7.46**	10.75	7.87	3.96
200	**9.71**	13.78	9.35	7.47
250	**14.57**	24.37	13.47	8.75
300	**23.11**	32.31	23.54	12.57
350	**29.70**	33.62	30.19	22.96
354	**30.02**	34.46	30.46	22.96



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). This allows to find the best approach for this specific problem. Possible algorithms are REINFORCE, truncated natural policy gradient (TNPG), Reward-Weighted Regression (RWR), Relative Entropy Policy Search (REPS), Trust Region Policy Optimization (TRPO), Cross Entropy Method (CEM), Covariance Matrix Adaption Evolution Strategy (CMA-ES) and Distributed Distributional Deterministic Policy Gradients (DD4G, Barth-Maron et al. 2018). The implementation of those different algorithms allows a future benchmarking for this or similar problems.
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.

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.