Report on Results

Deep Reinforcement Learning

Project 3: Collaboration and Competition

David Neuhäuser

1 Framework

In the current project, the goal was to train **two** agents which communicate with each other. In this context, an agent corresponds to a tennis racket which can bounce a ball over a net. A reward of $R_t = 0.1$ is provided to an agent for hitting the ball over the net, whereas an agent receives a reward of $R_t = -0.01$ for letting the ball hit the ground or hitting the ball out of bounds. Thus, in order to optimize the total score, the aim of the agents is to keep the ball in play for as many time steps as possible. The state space $\mathcal{S}$ has 8 dimensions corresponding to position and velocity of the ball and racket. Based on this information, the agent has to learn how to select actions in the best way. The action space $\mathcal{A}$ is continuous since two actions are available, corresponding to movement toward/away from the net, and jumping. The task is episodic, i.e. $t \in [0, 1, \dots, T]$.

In order to pass, the agents need an average score of 0.5 over 100 consecutive episodes (after taking the maximum over both agents). To be more precise, after each episode j, add up the rewards $R_t^{(i,j)}$ that each agent i received to get an individual score

$$G_{i,j} = \sum_{t=0}^{T} R_t^{(i,j)}$$

for each agent. This yields 2 scores $G_{1,j}, G_{2,j}$. Then, take the maximum

$$M_j = \max_i G_{i,j}$$

of these 2 scores leading to a single score M_j for each episode j. The environment is considered solved, when the average over 100 episodes of those average scores M_j is at least 0.5.

2 Detailed Description of the Algorithm

The algorithm which is used to train the agent for solving the task is the Multi-Agent Deep Deterministic Policy Gradient algorithm. It is one of the so-called Actor-Critic Methods (others are for instance TRPO, PPO, A3C or A2C) and uses **per agent** two neural networks for learning: one for the actor and one for the critic. In DDPG, the actor is responsible for the "policy part" which chooses the next action based on the current state. The critic is responsible for the "value part"which evaluates the action chosen by the actor. The general idea of the algorithm is to use value-based techniques in order to reduce variance of policy-based models. In MADDPG, we have a decentralized actor, centralized critic approach, i.e., the learned policies are only able to use local information at execution time. As the name of the algorithm already points out, there is no stochastic policy in the model.

In addition, some stabilizing techniques helped to improve the performance in training the model.

- it turned out that updating the weights only after every 15 steps and for every such step, updating the weights 5 times improved results a lot. Of course, these numbers could be seen as hyperparameters in the algorithm and could be tuned. Moreover, some other techniques contributed significantly towards stabilizing the training, too.

- experience replay: in order to de-correlate the state-action-reward-next state tuples $(s_t, a_t, r_{t+1}, s_{t+1})$, $(s_{t+1}, a_{t+1}, r_{t+2}, s_{t+2}), \ldots$, it is useful to create a replay buffer of certain size. Then sample at random from this buffer leading to (at least close to) i.i.d. samples instead of learning from tuples as they come over time. Note that in MADDPG, the buffer is a shared one.

- fixed Q-targets: fix the weights of the neural net in the target $\widehat{Q}$. Of course, this leads to the consequence that we need two neural networks for the actor and two networks for the critic, one as the target network and one as the online network, respectively. However, we can get rid of instability issues when updating the model weights.

- soft updates: contrary to the Deep Q-Networks algorithm where the target networks are updated by copying all the weights from the local networks after a certain number of epochs, we have two copies of weights for each network in the Deep Deterministic Policy Gradient framework: actor online, actor target, critic online, critic target. At every update step, we mix 0.1% of the online weights with the target weights to update the target weights of the actor and the critic.

- use of gradient clipping when training the critic network

An interesting topic in the provided solution is the underlying architecture of the neural networks for the actor and critic, respectively. More precisely, we investigated two different architectures which are as follows:

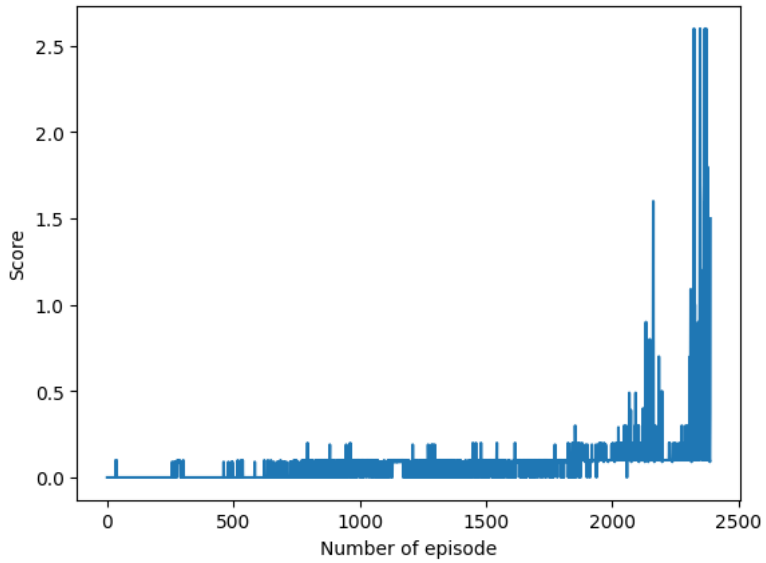
- Architecture 1:
  - Actor_2HL_BN: a neural net with two hidden layers. The second layer (first hidden) has 400 nodes, the third layer (second hidden) 300 nodes. Activation function is ReLU, except for the last one which is tanh. Additionally, we apply batch-normalization to the first hidden layer.
  - Critic_2HL_BN: a neural net with two hidden layers. The second layer (first hidden) 400 nodes, the third layer (second hidden) 300 nodes and finally the fourth layer 1 node. For details, we refer to the Python code. Activation function is ReLU, again. Additionally, we apply batch-normalization to the first hidden layers.
- - Architecture 2:
  - Actor_1HL: a neural net with one hidden layer. The second layer (first hidden) has 400 nodes. Activation function is ReLU, except for the last one which is tanh.
  - Critic_3HL_leaky: a neural net with three hidden layers. The second layer (first hidden) has 300 nodes, the third layer (second hidden) 200 nodes, the fourth layer (third hidden) 100 nodes and finally the fifth layer 1 node. For details, we refer to the Python code. Activation function here is leaky ReLU.

In the following table, all hyperparameters which were used for the Deep Deterministic Policy Gradient algorithm are listed:

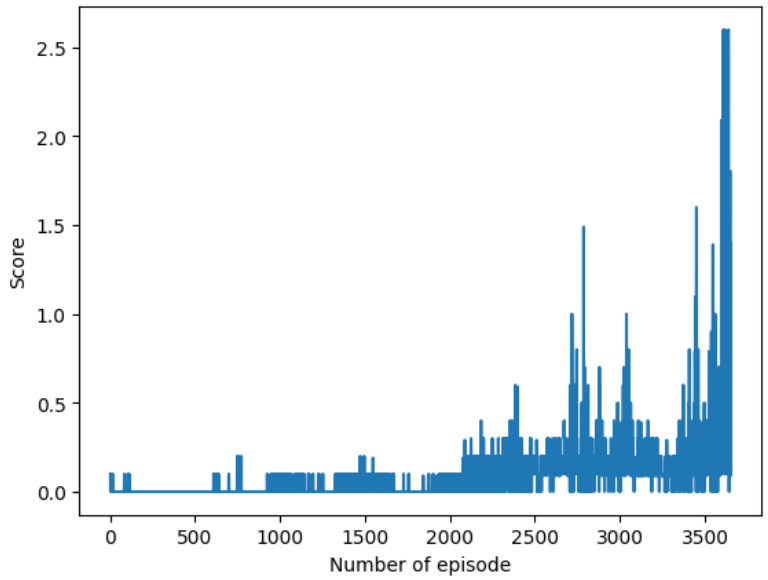
Hyperparameter	Value
replay buffer size	100000
batch size	256
discount factor γ	0.999
soft update parameter τ	0.001
learning rate actor	0.0001
learning rate critic	0.001
weight decay Adam optimizer	0.0
number of episodes	10000
maximum number of timesteps per episode	1000
negative slope leaky ReLU	0.01

3 Results

The performance of the actor-critic-agents via architecture 1 was as follows: The environment was solved in 2388 episodes with anverage score of 0.51 over the last 100 episodes, also see the following figure.



The performance of the actor-critic-agent via architecture 2 was as follows: The environment was solved in 3652 episodes with anverage score of 0.50 over the last 100 episodes, also see the following figure.



Obviously, architecture 1 effectively begins to learn from episode 1800 onwards. Contrary, architecture 2 starts to learn from episode 2100 onwards but seems to fail at episode 3300 in order to finally successfully learn until the end. There is a clear performance difference between architecture 1 and 2 with architecture 1 being the better one. As a consequence, at least minor efforts of changing the fundamental structure of the neural net seem to eventually bring performance increase. Beyond, we can consider the ideas of the next section in future.

4 Ideas for Future Work

- Usage of other network architectures
- Prioritized experience replay: instead of uniformly sampling from the replay buffer, there should be some kind of importance sampling, taking more relevant tuples with higher probability than minor important ones.
- Hyperparameter tuning
- Testing improvements of MADDPG or other algorithms than MADDGP: try for instance Mixed-Experience-MADDPG, see here.