Tennis Multi-Agent Project Report

Patrick Sowinski

This project implements a Deep Deterministic Policy Gradient agent (DDPG) to solve the cooperative Tennis environment from Unity ML Agents.
The project is part of the Udacity Nanodegree program on Deep Reinforcement Learning.

The Environment

The environment is implemented within Unity ML Agents. More info about these environments can be found here: https://github.com/Unity-Technologies/ml-agents

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 24 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, the agents must get an average score of +0.5 (over 100 consecutive episodes, after taking the maximum over both agents).

The Solution

The environment was solved with a Deep Deterministic Policy Gradient (DDPG) agent.
The agent has an “act” function, which determines the next action to take given the environment’s state. It also has a “step” function, which adds experiences to the replay buffer. This replay buffer is used for learning, which includes updating the network weights.
The agent uses an actor and a critic network, each of which has two versions, a local and a target network with the same architecture. So in total there are 4 networks. The target network weights are updated after every learning step with a soft update from the corresponding local network.

The actor networks each consist of 4 fully connected layers with ReLU activations between them (input and output layers + 2 hidden layers). The final layer output is fed into a tanh activation function to match the defined action value intervals from -1 to +1.
The input size corresponds to the size of the state space, given by the environment, which in this case has 24 dimensions. The output size of the networks corresponds to the number of dimensions in the action space, which is 2 here.
The hidden sizes of the layers are 512 and 256.

The critic networks each consist of 4 fully connected layers with ReLU activations between them (input and output layers + 2 hidden layers). The critic networks receive the environment’s state and the selected actions as inputs, which are fed into the network at different points. The state is fed into the first input layer directly. The actions are concatenated with the outputs of the first hidden layer (after ReLU activation).

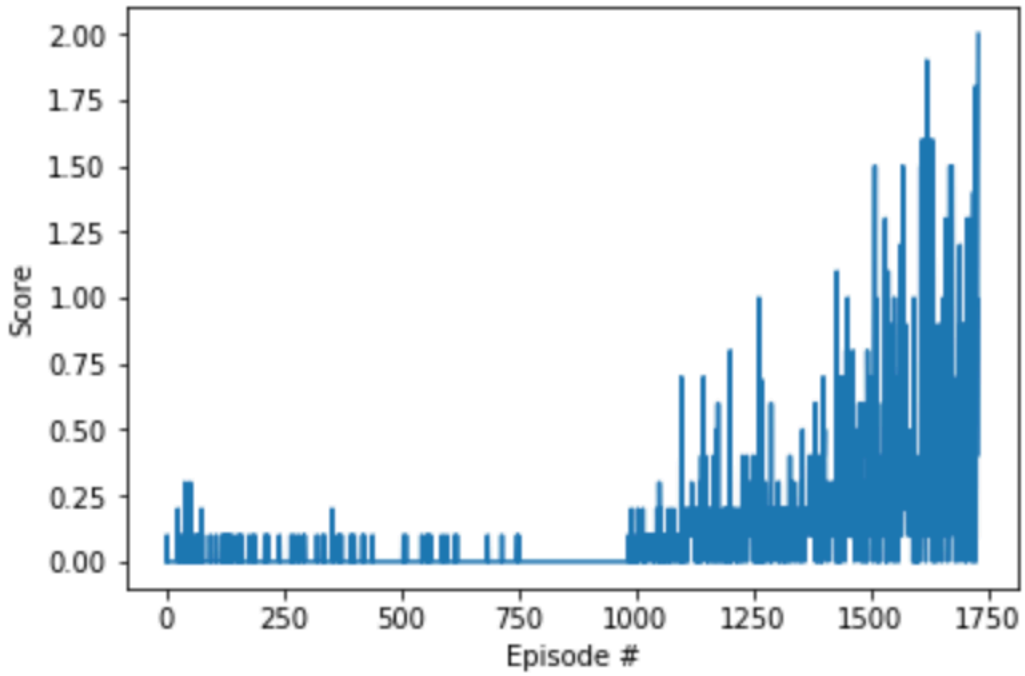
The input size corresponds to the size of the state space, given by the environment, which in this case has 24 dimensions. The output size of the network is 1, because it is only meant to approximate a scalar value. There is no activation function for the output layer.
The hidden sizes of the layers are 512 and 256. We need to remember that the 512 output values of the first hidden layer are concatenated with the given action values to form inputs for the second hidden layer with size 256.

The replay buffer has a very large size of 1,000,000, so it can store a variety of different experiences. A large replay buffer has shown to be important for this agent to learn in a stable way. This might be, because the actions for the Tennis game, in particular the timing of a jump, need to be very precise, which means we want to have a very detailed and extensive description of the state space available for learning. A larger buffer helps to achieve this.
For the same reason, we are using a rather large batch size of 256 here.
The factor gamma used for discounting rewards is set to 1.0, so it does not matter when a reward is collected.
The soft update factor tau is set to 0.05, which means it takes roughly 20 steps to completely overwrite the target network weights.
The learning rate for the actor network is set to 0.00005 and for the critic network it is 0.0001. We update the critic "more strongly", since we want to have a good feeling for the value of some actions before changing the actions we are making.

Results

Below is the plot of rewards per episode during training. The environment was solved as soon as the agent reached an average reward of +0.5 (5 hits over the net) throughout the previous 100 episodes.
The score per episode has consistently exceeded +30 after just 30 episodes. The average in episodes 9 to 108 was above the required threshold.

```
Episode 100     Average Score: 0.02
Episode 200     Average Score: 0.02
Episode 300     Average Score: 0.01
Episode 400     Average Score: 0.01
Episode 500     Average Score: 0.00
Episode 600     Average Score: 0.01
Episode 700     Average Score: 0.00
Episode 800     Average Score: 0.00
Episode 900     Average Score: 0.00
Episode 1000    Average Score: 0.01
Episode 1100    Average Score: 0.08
Episode 1200    Average Score: 0.12
Episode 1300    Average Score: 0.15
Episode 1400    Average Score: 0.16
Episode 1500    Average Score: 0.26
Episode 1600    Average Score: 0.34
Episode 1700    Average Score: 0.44
Episode 1729    Average Score: 0.50
Environment solved in 1629 episodes!    Average Score: 0.50
```



You can also watch the trained Tennis agent here:
https://www.youtube.com/watch?v=eK3YFpLWvS8
In the video, which is sped up to 4x speed, the two players hit the ball back and forth indefinitely. This can go on for many minutes, if not longer.
This indefinite type of play does not occur in every episode. Sometimes the agent only hits the ball one or two times and then the ball goes out of bounds. It seems like the agent first needs to maneuver the ball towards a more stable playing position, which it does not always succeed to do. Once it reaches this stable position, where the ball always lands roughly ¾ of the way from the net to the end of the field (as seen in the video), it can play in this position for a very long time.
We can also see in the graph that there is a large variance between episodes, where sometimes the score is close to zero and sometimes it corresponds to 20 hits over the net or more (in which case the time limit for episode comes into play).

Ideas for future work

One possible improvement would be to use prioritized experience replay, so we learn more often from uncommon experiences that provide a lot of information.

We used a decaying factor epsilon to select random actions to make sure the agents are not too greedy at the beginning of training. This improved the training results by a lot and prevented the agents from being stuck in a corner of the playing field.
We could improve this even further by adding an exploration period of 1000 episodes before we start learning, so we have a more complete representation of the state space before we modify network weights.
The same could also be achieved with a very large batch size, since we do not train before our replay buffer has reached the required batch size. However, this would increase computation times significantly, which would not be feasible in the long run.

The next step for this environment would be to add movement of the ball and players in a third dimensions, i.e. along the depth of the playing field. This would make the task even more complex.

It would also be interesting to see if the players can handle multiple balls on the playing field at once.