Project 3: Multi Agents

Featuring the Tennis environment

Learning Algorithm

Deep Deterministic Policy Gradient (DDPG)

Deep-Deterministic-Policy-Gradient is an Actor-Critic algorithm which is model-free as well as off-policy (Lillicrap et. Al. 2015). The method consists of two main components. The Actor and the Critic. Both are represented by Neural Networks. It uses concepts from both the Deep-Q-Network (DQN) as well as policy methods. By combining elements of both, DDPQ (or actor-critic models in general) are able to apply the concepts of DQNs on continuous action spaces through application of policy evaluation and noise, while also improving the stability and robustness of policy methods.

The Actor is represented by a neural network (NN). It is used to evaluate the best action to take considering the current state of the environment, as calculated by the current policy.

The Critic is also represented by a NN and is used to improve the learning of the actor. To accomplish this, the critic is also trained on the same pool of data as the actor but is then used to calculate the loss of the current actor model. This improves the trajectory of the learning.

Parameters:

The parameters were set as following:

BUFFER_SIZE:	The size of the experience buffer (deletes from bottom if overflow). Seemed fairly robust to changes.
BATCH_SIZE	The size of the experience sample the networks are updated with. Best Value was 128. 256 reduced training velocity significantly.
random_seed	A random seed to create reproducible results. This greatly had effect on the performance. Selected seed was 1.

GAMMA	The cost of future rewards. Did not seem to impact greatly except for fairly high/low values. Was left to 0.99.
TAU	The gravity of change to the target network during update. This was set to 0.001. Higher values led to bad performance.
LR_ACTOR	The Learning Rate of the critic.
LR_CRITIC	The learning rate of the critic. Positive Effect if it was lower than the LR of the Actor
WEIGHT_DECAY	Decay of the network weights. Was not used in this sample.
UPDATE_EVERY	Update Cycle of multi-agents. Was not used in this sample.
UPDATE_NETWORK	Update Number of multi-agents. Was not used in this sample.
max_t	The number of steps taken every episode. Performance dropped if under 600 or over 1200.

Components:

Environment:
This is the environment the agent interacts with. In this case the tennis environment from the UnityAgents. The environment is reset at every episode and the actions calculated by the agent are passed in order to make a step in time. Each environment consist of two agents which have independent actions and observations.

Memory:
This is the shared memory buffer. If the environment only uses one agent, this is identical to an agent owned memory. If the environment uses multiple agents, this memory serves as shared experience pool for all agents.

Agent:
This is the instance of the Agent class that interacts with the environment. At the beginning of every episode the agent is reset. It then calculates the preferred action with the .act() function, using its local actor network. After interaction with the environment, the interaction is passed to the agents.step() function which saves the experience and eventually updates the local and target networks.

Scores, scores_deque, scores_list, data:
These variables are used to store the performance of the agent over time.

Model:

The model used for both the actor and critic are neural nets with three layers.
The actor is a very simple net that uses three fully connected layers. The input layer converts the state of 33 values to a net of 128 neurons. The second layer processes this to 256

units/neurons. The output is then converted to the action size of 4 in the output layer. All layers are activated with the RELU function, except the output layer, which is activated by tanh.

The model used for the critic is similar in structure, as it uses the same layers, although the output layer only has a size of one. Also, in the processing, the first layer is converted to a categorical value to atone for the critics eval. Output, which eliminates the need for the tanh activation of the output layer.

Both Critic and Actor are comprised of a local model, which is trained directly, and a target model which is updated using a soft-update method.

Training

Experiments:

In the following section I go into detail about my experiments and which changes I made to evaluate and improve performance of the model.

First experiment:

In the first experiment I adapted my Code from the second project and modified it to work with multiple agents. This was somewhat simple, as it mostly meant adding a few loops for e.g. the agent.act() and agent.step() functions. A bit more challenging was the adaptation of the neural network, as the new input was one-dimensional, but the nn.Batchnorm1D function requires 2D or 3D data. With a bit more effort than I care to admit I found that transforming the state data from an array to a tensor and then using unsqueeze brought the data in a working format. I was worried that this would hinder training, but after a short run I could see some learning, more or less.

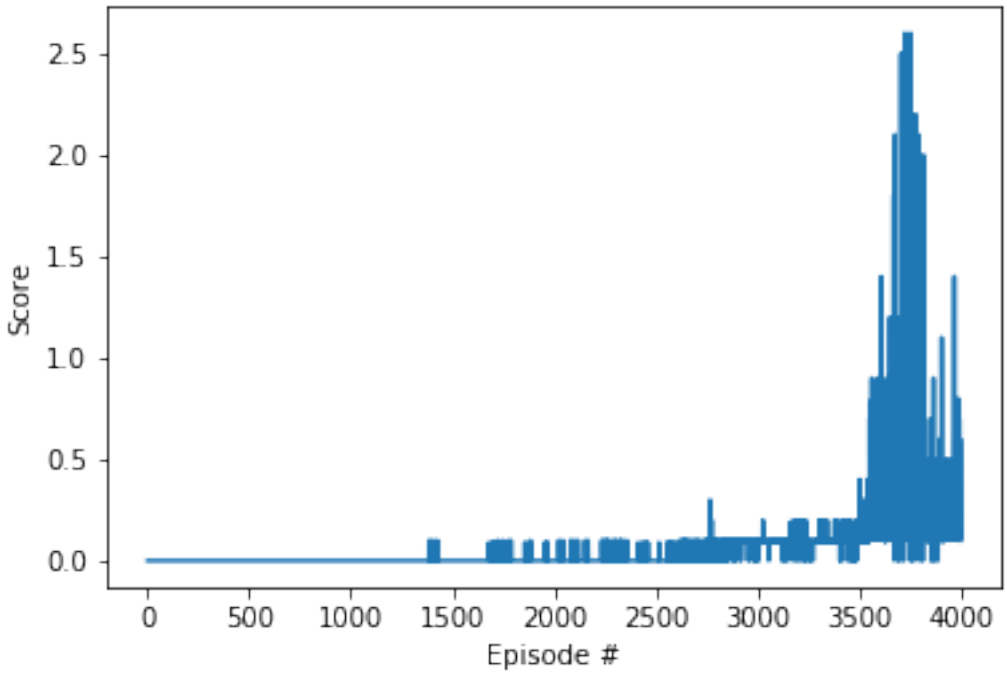
```
state = torch.from_numpy(state).float().to(device)
state = state.unsqueeze(0)
```

Second experiment:

- Old algorithm
- Externalized actor target

After confirming that the agent was actually learning, I externalized the actor target net, so that both agents would use and update the same target net. As there were only two agents, I didn't worry much about stability with two updates every step. I let the agent run for 5.000 episodes and was able to reach the target at about 3.500 episodes. Although far from optimal, this was a good indication, that I was headed in the right direction. The agent was fairly stable, but learning was super slow. As I was still mostly using the parameters optimized for the reacher environment, I decided to next run at max 2.500 episodes and optimize for faster training. I noticed, that with increasing performance, the agents obviously played longer and

the episodes took longer to



finish.

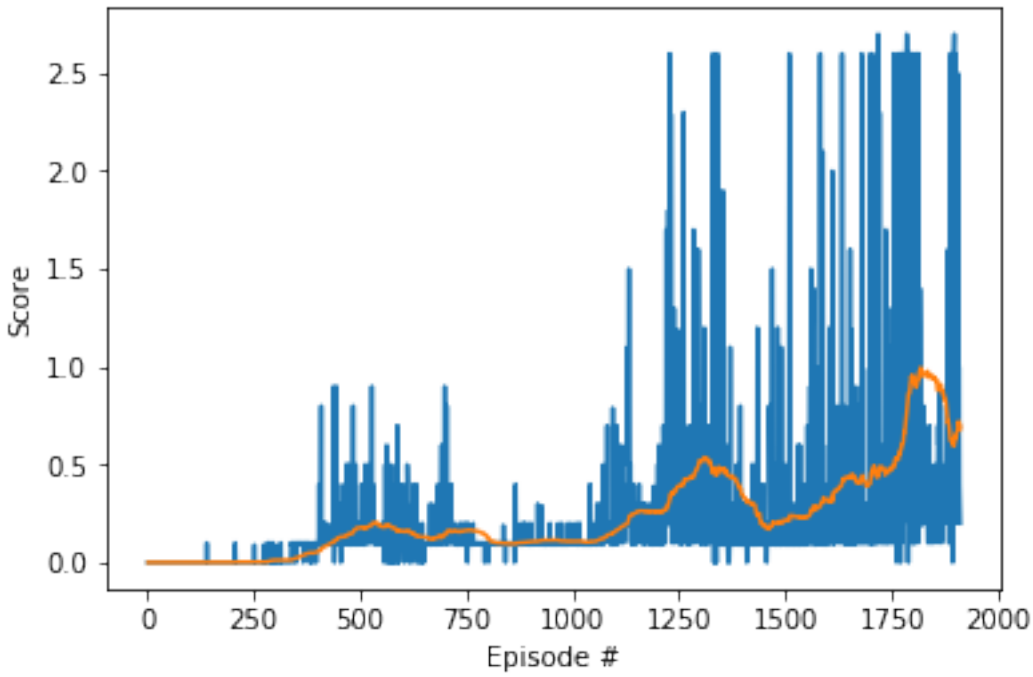
Third experiment:

- Reduced training to 2500 episodes
- Gamma 0.99 → 0.999
- Batch 128 → 256
- Tau 0.0001 → 0.001
- Lr critic 2e-4 → 2e-3

The third experiment brought quite a few changes to the parameters. Firstly, as the speed aspect of the environment is not super important, I reduced gamma to 0.999. Rather than fast ball exchanges, I think that “safe” plays would benefit learning more.

Secondly, I increased the batch size, as it seemed a bit small with 128, compared to the amount of combinations and experiences. The two most important changes were the increase of the critic learning rate and the tau. As the agents were learning very slowly, I increased the learning rate to speed up training and hoped to not get completely unstable results. I did not increase actor learning rate, as was it was being updated twice per step anyway.

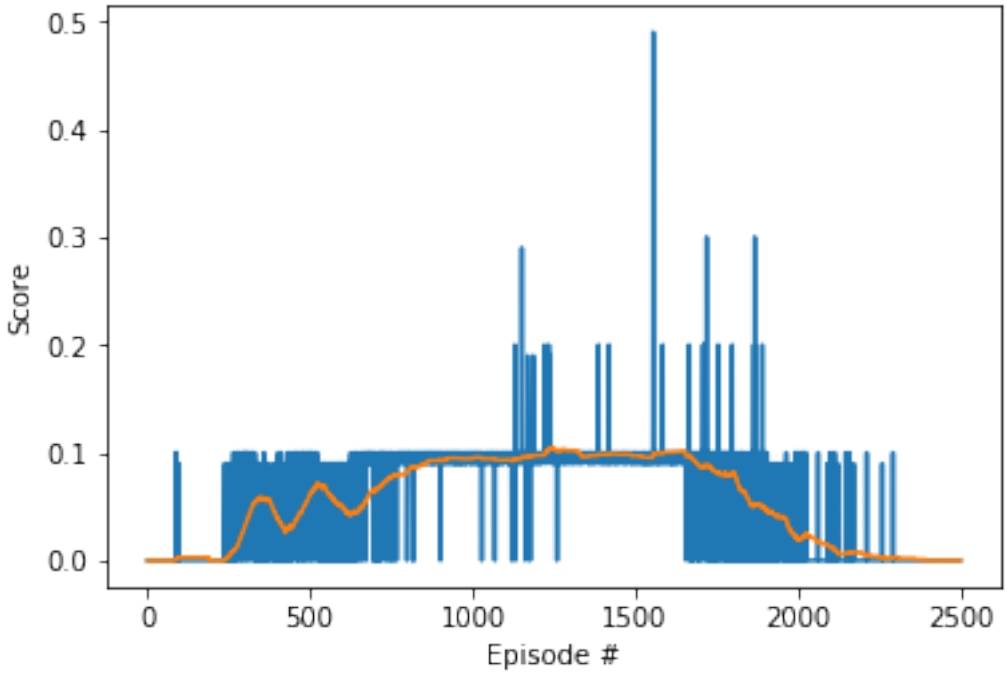
The results were very positive, as the agent learned fairly quickly and was able to reach the target in around 1.000 episodes. Still, performance was not super stable, so that was the next challenge to overcome.



Fourth experiment:

- Externalized target critic

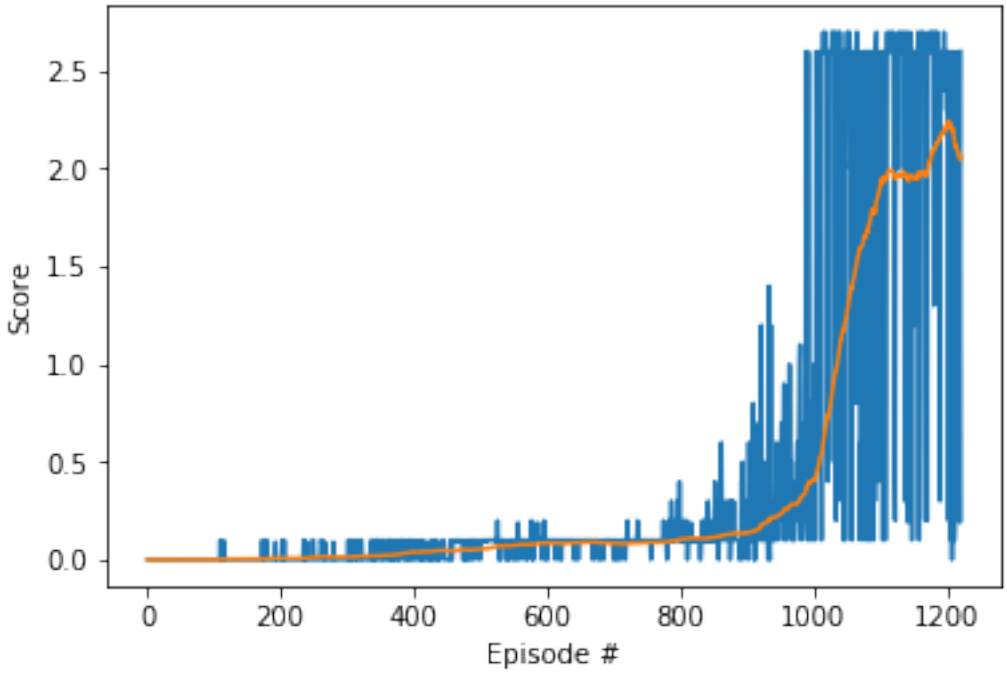
In the next experiment I updated my data displays, in order to faster evaluate training without having to run the whole 2.500 episodes.
My thought was, that externalizing the critic may improve the stability, as the critic would be more robust to different situations. After running the experiment, I saw that this was not the case. The Agents performed very poorly. Although performance increase more rapidly than in the other tests, it failed to reach relevant scores and made a drop to around 0, from which it did not recover.



Fifth experiment:

- Internalized target critic
- Added dropout 0.2 after second hidden layer of actor

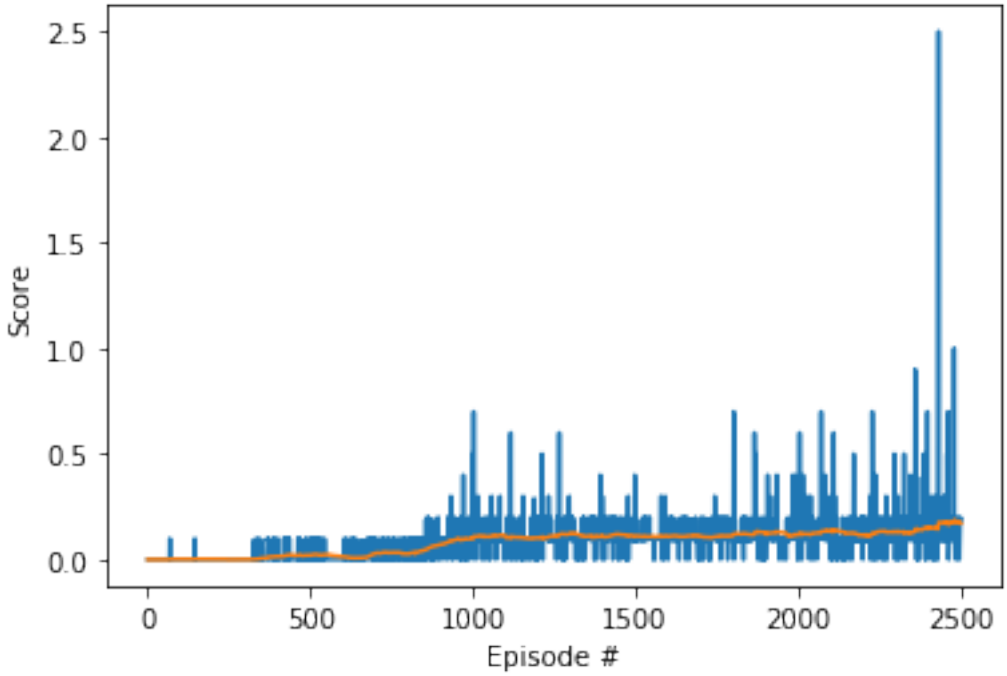
In my fifth experiment I reverted the externalization of the critic. This did not seem a very promising path. I still wanted to improve stability, so I chose to add a dropout layer of 0.2 after the second hidden layer of the actor model. I refrained from adding it to the critic in order to evaluate the difference more isolated.



Sixth experiment:

- Added dropout 0.2 after second hidden layer of critic

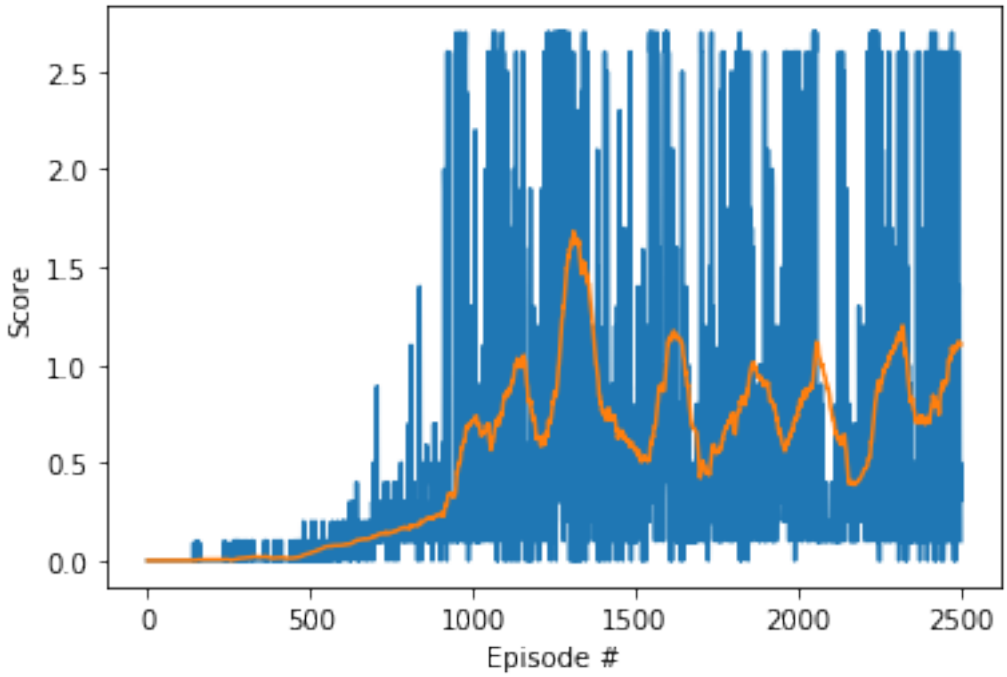
In high hopes after the effects of the dropout layer in the fifth experiment, I also added one to the Critic. This had disastrous effects on the performance. The model was not able to reach performance anywhere near the goal. The learning did seem to be stable, but it was unbearably slow. This was obviously not beneficial.



Seventh Experiment:

- Removed drop-out from critic
- Let run for 2500 episodes

In my seventh and last experiment, I removed the drop-out layer from the critic and let the algorithm run for 2500 episodes. We can see, that there are regular drops in performance, but the model recovers too. Peaks reach performance of up to 1.8 on average. When watching the algorithm play, the performance is very good. Satisfied with the results, this is my final submission.

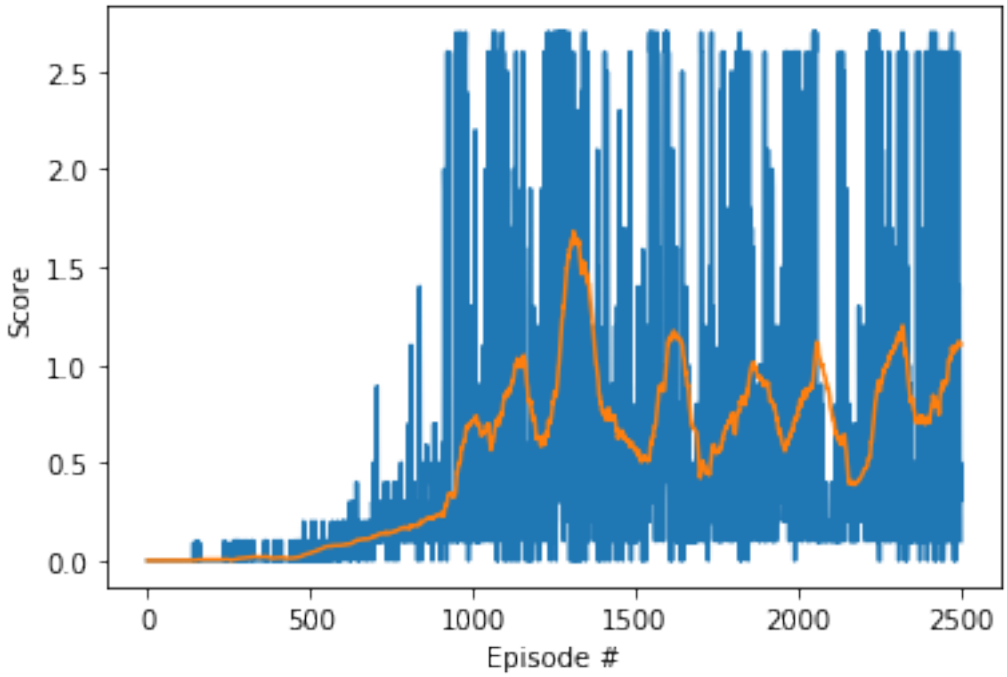


Performance:

The final Agent shows slow training at the beginning which ramps up after around 500 episodes, reach scores of 0.1 – 0.2 on average. Afterwards the performance increases quickly, leading to performances of around 0.5 after about 1.000 episodes. The algorithm peaks at around 1300 episodes with occasional drops, reaching a performance of about 1.7 on average over the last 100 episodes. After the peak, the performance sways between 0.5 and 1.2.

Plot of rewards:

Here we can see the plot of rewards in raw and aggregated over 100 episodes for evaluation.



Ideas for Future Work

This environment does require many episodes to complete. The computation time is very long and only improving a bit on GPUs. I think that A3C might provide even better performance, as the algorithm benefited greatly from externalizing the actor model. The parallel optimization could also cut computation time and training episodes per agent. Having a more diverse mix of agents would also help improve learning stability and robustness.

References:

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., ... & Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, O. P., & Mordatch, I. (2017). Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems* (pp. 6379-6390).