THE TENNIS ENVIRONMENT SOLVED USING A DEEP DETERMINISTIC POLICY GRADIENT ALGORITHM

JULIAN WERGIELUK

ABSTRACT. This short note provides a concise description of the model architecture and learning algorithms of the agent developed in this project. We also report learning performance of the agent and provide a list of possible future model improvements.

1. DESCRIPTION OF THE LEARNING ALGORITHM

For this problem, we use a standard Deep Deterministic Policy Gradient algorithm (DDPG) and parameterize it in a way to make it suitable for multi-agent environments.

Deep deterministic policy gradient algorithm is an actor-critic type method with a deterministic actor policy π mapping states from $\mathbb{S}$ to actions in $\mathbb{A}$. During the training, the action-value function Q is optimized using a standard DQN algorithm as described in [2] employing a replay buffer and a separate target network.

The policy network is trained using a policy gradient derived from the action-value function approximation. Specifically, the parameter vector θ is given as $\theta = (\theta^Q, \theta^\pi)$, where θ^Q determines Q and θ^π determines π. Let $J(\theta)$ be the cumulative expected reward. Then the policy gradient can be approximated by

$$\nabla_{\theta^\pi} J \approx \mathbb{E}\left[\nabla_a Q(S_t, \pi(S_t)) \nabla_{\theta^\pi} \pi(S_t)\right].$$

To improve the algorithm stability we use soft target updates for the parameters of both actor and critic networks, described in [2], and batch normalization.

2. TRAINING ANALYSIS

We train a DDPG agent for 1900 episodes consisting of at most 1000 time steps each. The cumulative reward averaged over 100 episodes reaches the level of 0.5017 after completing the episode 494.

3. ALGORITHM PARAMETRIZATION

The Actor network architecture. The actor is a feed-forward neural network mapping the state of the environment to a deterministic action. The network consists of three layers sandwiched with

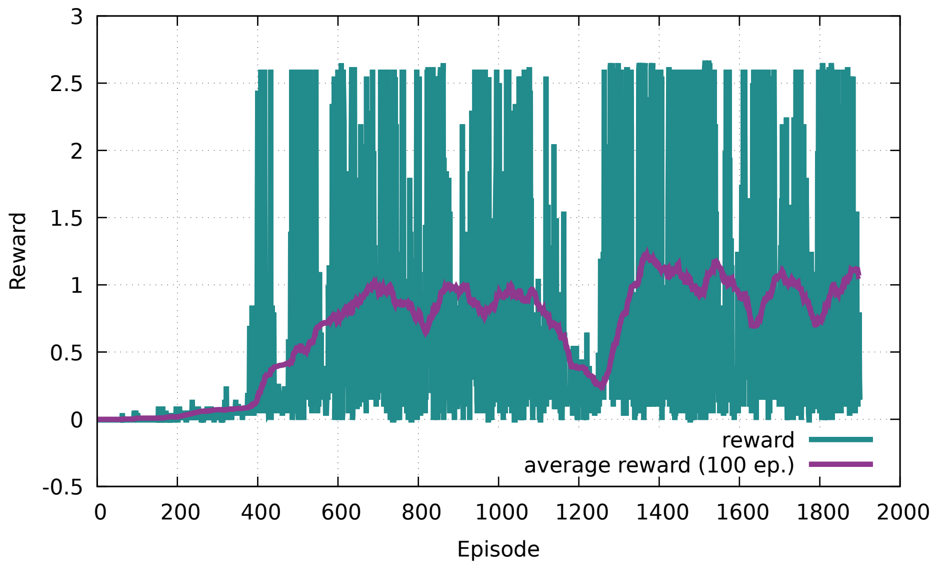

FIGURE 1. We train a DDPG agent for 1900 episodes. The "reward" curve gives the maximum of individual agents rewards at the end of each episode. The "average reward" curve is the rolling mean of the "reward" curve calculated over 100 episodes. (For episodes $1,\cdots,99$ we use an expanding window.)

batch normalization. The final layer uses the tanh nonlinearity to produce an action vector in the cube $[-1,1]^2$.

```
Actor(
  (_pi_net): Sequential(
    (0): BatchNorm1d(24)
    (1): Linear(in_features=24, out_features=128)
    (2): BatchNorm1d(128)
    (3): ReLU()
    (4): Linear(in_features=128, out_features=128)
    (5): BatchNorm1d(128)
    (6): ReLU()
    (7): Linear(in_features=128, out_features=2)
    (8): Tanh()
  )
)
```

The Critic network architecture. The critic is a feed-forward neural network approximating the state-action value function Q mapping state-action pairs to the expected cumulative reward. The critic network consists of two subnetworks: a one-layer network combined with batch normalization that maps the state to a high-dimensional state representation, and a two-layer network that

TABLE 1. List of hyperparameters and their values

Hyperparameter	Variable name	Value
Replay buffer size	`BUFFER_SIZE`	1e6
Batch size	`BATCH_SIZE`	128
Discount factor (γ)	`GAMMA`	0.99
Tau	`TAU`	1e-3
Actor learning rate	`LR_ACTOR`	0.001
Critic learning rate	`LR_CRITIC`	0.001
Initial value of the OU process scaling factor ε	`EPSILON`	1.0
OU mean reversion level		0.0
OU mean reversion speed		0.15
OU volatility (sigma)		0.2
Learning frequency		every 5 time steps

maps the aforementioned representation of the state concatenated with the action vector to the approximation of the state-action value $Q(s,a)$.

```
Critic(
  (_q_state_net): Sequential(
    (0): BatchNorm1d(24)
    (1): Linear(in_features=33, out_features=128)
    (2): ReLU()
  )
  (_q_net): Sequential(
    (0): Linear(in_features=130, out_features=128)
    (1): ReLU()
    (2): Linear(in_features=128, out_features=2)
  )
)
```

The list of hyperparameters used by the agent is listed in Table 1.

4. IDEAS FOR FUTURE WORK

- Use of a real multi-agent reinforcement learning algorithm explicitly taking the non-stationarity of the environment into the account.
- Actor and critic could use a common network for state processing.

REFERENCES

[1] Timothy P. Lillicrap et al. *Continuous control with deep reinforcement learning*. 2015. arXiv: 1509.02971 [cs.LG].

[2] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *Nature* 518.7540 (Feb. 2015), pp. 529–533. ISSN: 00280836.

Email address: `julian.wergieluk@risklab.com`