P3_Collaboration and Competition: Tennis with DDPG

Ashwati Das

Introduction *(From Udacity Course Website)*

The goal of this task is to play collaborative tennis, i.e., to keep the game going for as long as possible. As such, the reward framework is as follows:

- +0.1 – if agent hits ball over the net
- -0.1 – if ball hits ground or goes out of bounds

The state space consists of 24 states including the ball's and racket's position and velocity. Each agent sees this state space locally. The action space consists of 2 continuous actions:

- Toward and away from the net
- Jumping

In this episodic task, the agent is considered successful if an average score of +0.5 over 100 consecutive episodes is achieved and maintained (taking max over both agents).

Learning Algorithm

This task is approached by blending lessons learned from traditional value and policy-based reinforcement learning methods with Artificial Neural Networks, leading to a Deep Reinforcement Learning regime. Similar to the DQN network employed in P1: Navigation, the DDPG algorithm also employs an off-policy training with random samples drawn from a replay buffer to improve the stability of training. DDPG differs from DQN by employing the local and target networks for an actor and critic network each. The actor network is responsible for deterministic predictions on actions to take from a particular state, and the critic network evaluates the quality of these actions by computing the action-values (Q-values). Some noise is also incorporated via the Ornstein-Uhlenbeck process to aid with training. Detailed information on this strategy is presented by Lillicrap et al. [1] and Yoon [2]. Fig. 1 illustrates the associated pseudo-code implementation. A shared memory buffer between the two agents is also incorporated to aid in achieving collaboration between them.

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q, \theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

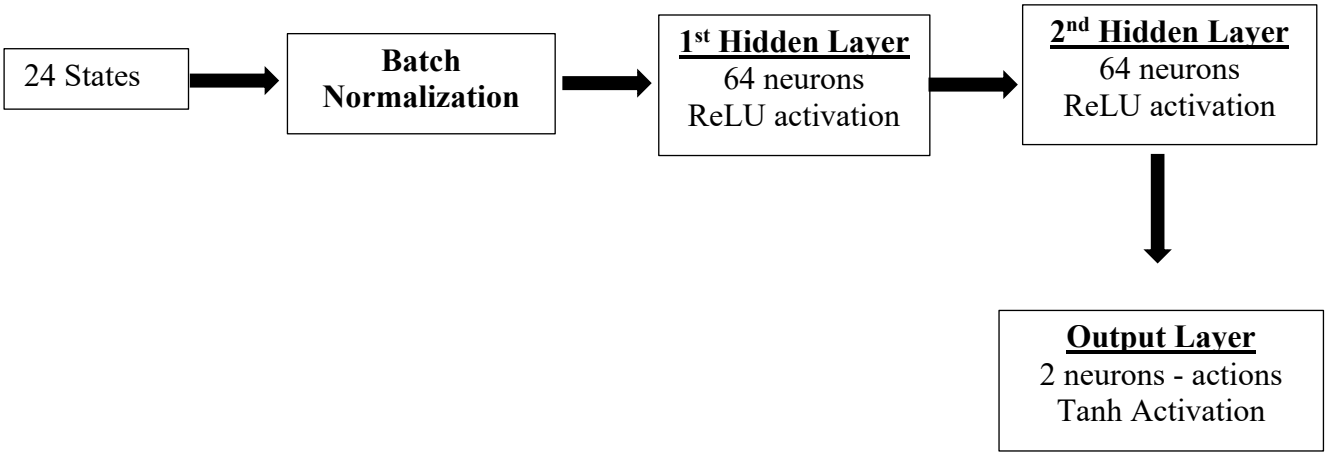
Figure 1: DDPG Algorithm (Credit: https://towardsdatascience.com/deep-deterministic-policy-gradients-explained-2d94655a9b7b)

Implementation

The implementation of the algorithm consists of design choices associated with the layer type, neurons and algorithms in the neural networks, and the hyperparameters for tuning these networks and for the training process.

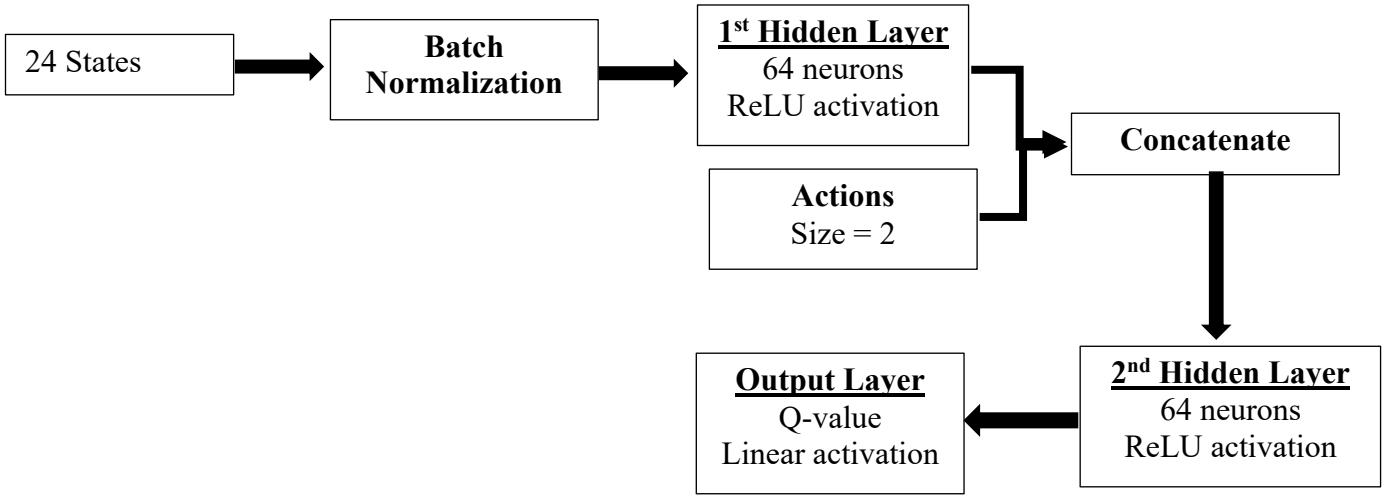
Neural Network Architecture – Actor Network

The actor neural network takes in the states as inputs and outputs the action to be taken from the state via a deterministic policy. This network consists of fully connected layers with the following neuron count and activation functions. Note that batch normalization is employed to address the numerical challenges arising from the variables in this training process being on different scales (e.g., positions, velocities).



Neural Network Architecture – Critic Network

The critic neural network consists of fully connected layers with the following neuron count and activation functions. Note that batch normalization is employed to address the numerical challenges arising from the variables in this training process being on different scales (e.g., positions, velocities). The gradient norm associated with this network is also clipped to aid with stability during training.



Hyperparameters

The training outcomes are very sensitive to the hyperparameters – thus, appropriate tuning is key to the agent being successful in its task. The following are the parameters used for the training process:

Parameter	**Value**
Number of episodes	2000
Number of steps within an episode	500000
Random Seed	0

The following are the hyperparameters used to tune the reinforcement learning process:

Hyperparameter	**Value**
Replay Buffer Size	1e5
Batch Size	512
Discount Factor (γ)	0.99
Soft Update Parameter (τ)	1e-3
Learning Rate - Actor	1e-3
Learning Rate - Critic	1e-3
Weight Decay	0
OuNoise - Asymptotic Mean – μ [3]	0
OuNoise - Strength of Reaction to Perturbation – θ [3]	0.15
OuNoise - Variation of the Noise – σ [3]	0.2

The Adam algorithm is employed for the gradient updates associated with the neural network computations.

Files

Tennis.ipynb – Main file consisting of the environment set-up
Constant_params.py – File consisting of constant values used in the analysis
ddpg_agent.py – File consisting of the definitions for the 'agent', abd 'OuNoise' classes,
ddpg_multiAgent.py - File consisting of the definitions for the 'MultiAgent', abd 'ReplayBuffer' classes
model.py – File consisting of the neural network architectures and choices

Plot of Rewards

Using the parameters and hyperparameter choices listed earlier, the environment is solved (average reward of 0.5 over 100 consecutive episodes) in 1326 episodes, as displayed in Fig. 2.

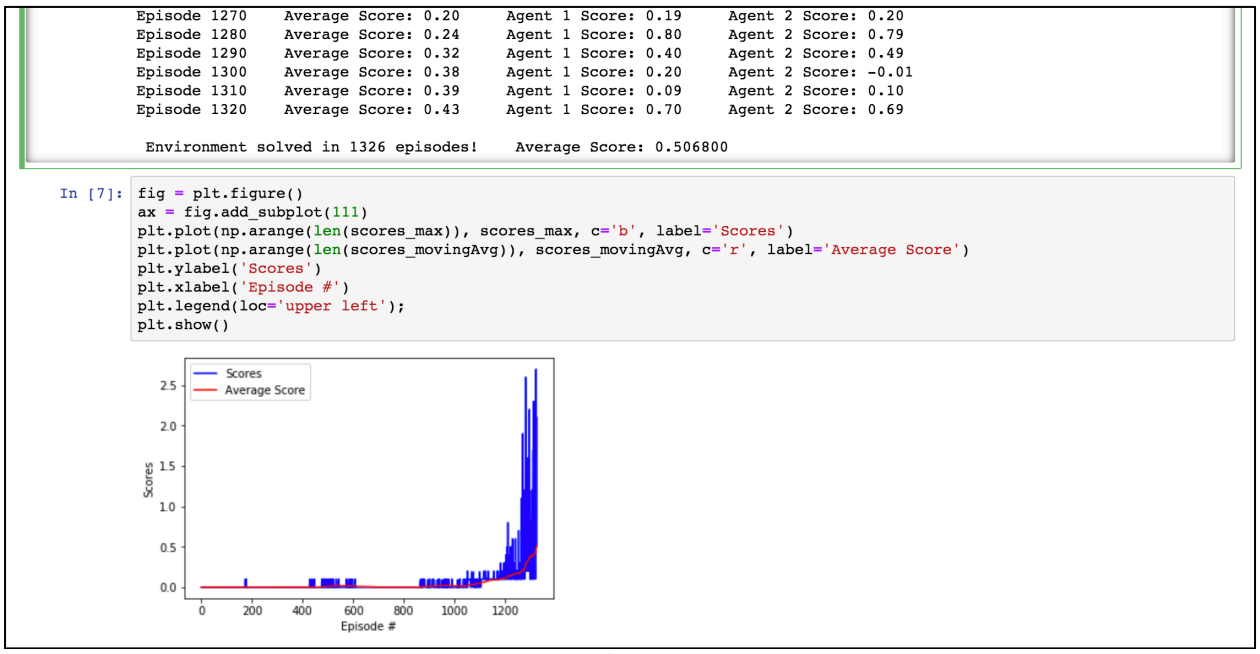


Figure 2: Plot of scores over episodes

Ideas for Future Work

- Further tuning of the hyperparameters would be attempted to investigate the potential impact on the results
- The impact of having separate replay buffers for the two agents – although this may not lead to the desired cooperative tennis playing
- Modification of the neural network architecture choices could also be investigated
- Incorporating prioritized experience replay could also be helpful
- Investigate outcome of employing MADDPG algorithm where critic network incorporates all agents' state and action information [4].

References

1. T. Lillicrap et al., *Continuous Control with Deep Reinforcement Learning*, ICLR, 2016, arXiv:1509.02971v6
2. Yoon, *Deep Deterministic Policy Gradients Explained*, https://towardsdatascience.com/deep-deterministic-policy-gradients-explained-2d94655a9b7b
3. http://web.math.ku.dk/~susanne/StatDiff/Overheads1b
4. R. Lowe et al., *Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments,* arXiv:1706.02275v3