Tennis Project

Project Description

The project consists of solving the Tennis environment which consists of two agents that hit a ball over the next. The task is episodic.

- State Space: consists of 8 continuous variables corresponding to the position and velocity of the ball and the racket

- Action space: consists of 2 continuous variables between -1 and 1 corresponding to the movement toward (or away from) the net, and jumping

- Reward: The agent receives a reward of +0.1 when he hits the ball over the next and -0.1 when he lets the ball hits the ground or hits the ball out of bounds.

The score is computed at the end of each episode by taking the max of the rewards accumulated by each agent. The environment is considered solved when the average score is higher than 0.5 on average for 100 episodes.

Implementation Description

Since the action space is continuous, it seems wise to use the Deep Deterministic Policy Gradient method as it is more adequate for a continuous action output.
Deep Deterministic Policy Gradient is composed of an actor that outputs an action using the state as inputs and a critic that outputs an "evaluation of the future expected rewards" using the state and action as input. Both are updated using a separate target that gives the architecture more stability, since the target is updated through soft-updates.
Deep Deterministic Policy Gradient introduces the use of a replay buffer to

sample experiences from it randomly to avoid having correlated observations when training, hence invalidating the independent and identically distributed samples hypothesis.
The Deep Deterministic Policy Gradient paper suggests using Batch Normalization technique as well to scale input features, I used it to scale the state vector after the first layer before its concatenation with the action vector.
Deep Deterministic Policy Gradient is an off-policy algorithm that explores actions by adding a noise to the policy output. In the algorithm paper Ornstein-Uhlenbeck process was used, I used the same noise in my implementation.

Learning Algorithm

The agent is composed of an actor and a critic. At each learning step, the next action is computed by inputting the next state to the target network of the actor. The critic target network takes as input the next action and the next state and outputs the expected future rewards. By summing the current reward and the discounted future rewards, we get the target Q-value. Using the output value by the critic network, we can optimise the critic network loss. The equations representing this process are the following:

$$next\_action = actor\_target\_net(next\_state)$$

$$Q_{target} = rewards(state, action) + \gamma \cdot critic\_target\_net(next\_state, next\_action) \cdot (1 - done)\ Q_{expected} = critic\_net(state, action)$$

$$w_{critic} = {}^{min}(Q_{target} - Q_{expected})^2\ w$$

As for the actor network, the predicted action is computed using the state.
The actor network loss is the negative expected future rewards computed by the critic network.

$$predicted\_action = actor\_net(state)$$

$$w_{actor} = {}^{max}critic\_net(state, predicted\_action)\ w$$

The actor target and the critic target networks are updated using a soft-update process. The target networks weights are updated at each learning step using the following formula:

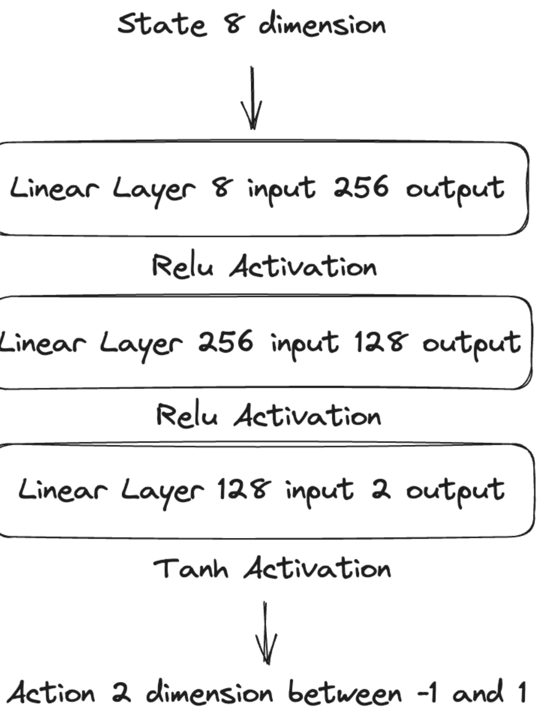
$$w_{-} = \tau \cdot w + (1 - \tau) \cdot w_{-}$$

To comply with the fixed target logic, τ is small (the value 3.10^{-1} is used in this project). Since, future reward should count less then present rewards, we use a discount factor γ of 0.99.

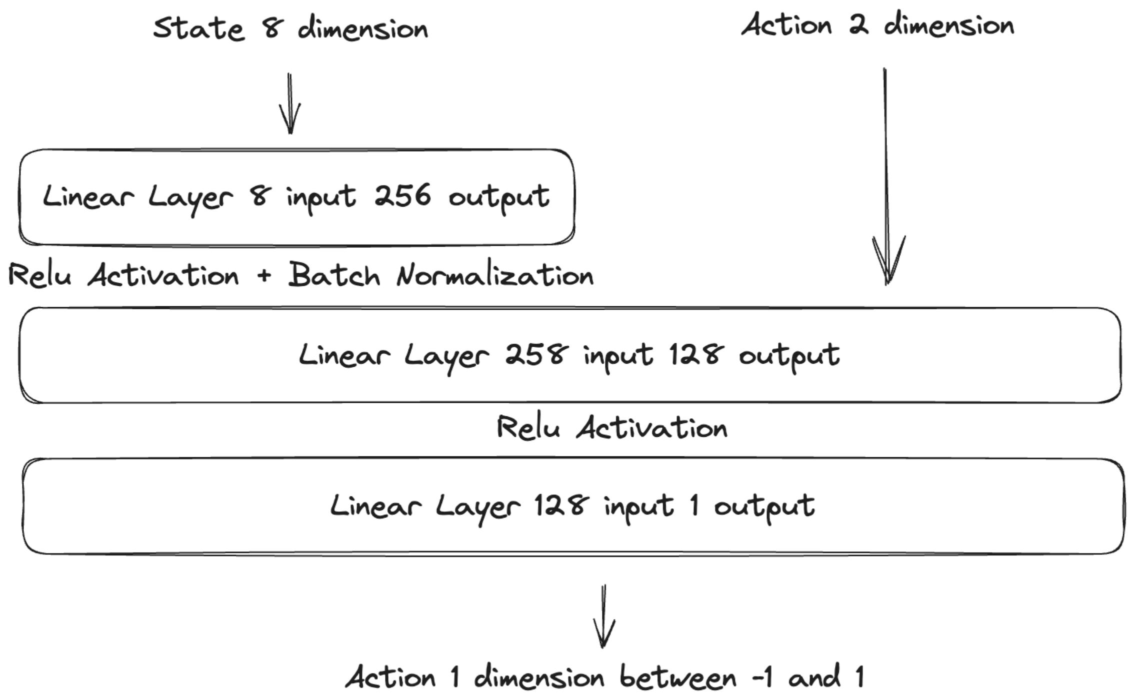
Last but not least, since this is an off-policy method we choose a decaying noise using Ornstein-Uhlenbeck process with a sigma decaying factor of 0.9 and a max noise of 0.2.

The optimizer of the actor and critic networks is Adam with a learning of 10^{-4} both. For training we sample a batch of 512 experiences from a buffer keeping the last 10^{5} experience tuples.

Actor architecture

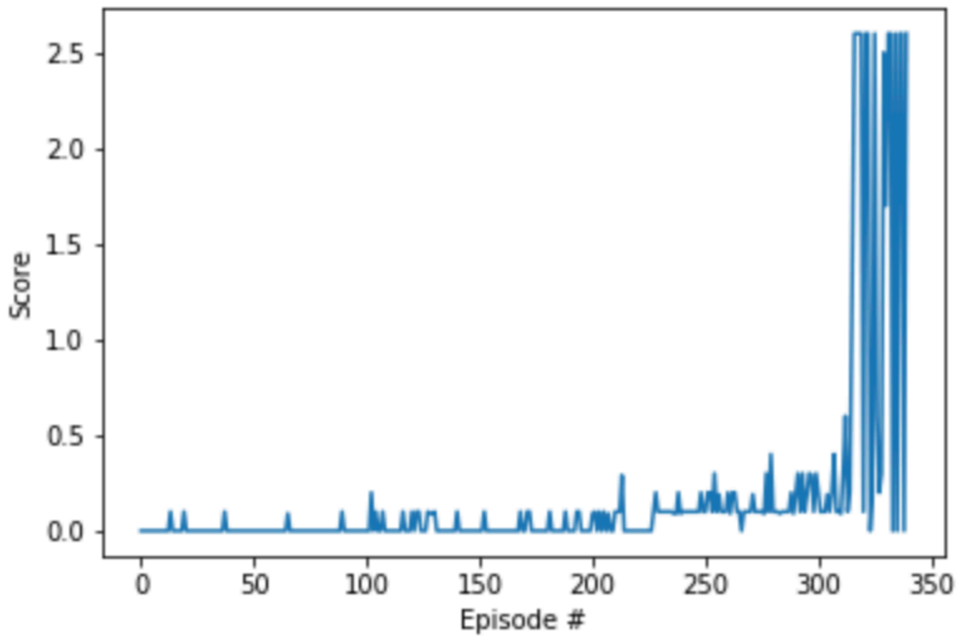

State 8 dimension
Linear Layer 8 input 256 output
Relu Activation
Linear Layer 256 input 128 output
Relu Activation
Linear Layer 128 input 2 output
Tanh Activation
Action 2 dimension between -1 and 1

Critic architecture


State 8 dimension
Action 2 dimension
Linear Layer 8 input 256 output
Relu Activation + Batch Normalization
Linear Layer 258 input 128 output
Relu Activation
Linear Layer 128 input 1 output
Action 1 dimension between -1 and 1

Rewards Plot

The environment was solved in 340 episodes by reaching a score of +0.52 over the last 100 episodes.



Ideas for Future Work

- Experiment different noise processes effects on the learning curve such as gaussian process
- Test hard-update instead of soft-update for the target networks by updating the networks only after a certain number of time steps
- Test different Actor and Critic networks architectures with different weights initialization techniques
- Perform a random search for the agent hyperparameters for a better score and convergence

Resources:

Reacher Project

DDPG Paper: https://arxiv.org/abs/1509.02971

https://medium.com/geekculture/policy-based-methods-for-a-continuous-action-space-7b5ecffac43a

https://github.com/vy007vikas/PyTorch-ActorCriticRL/tree/master