DRL for Tennis

1. Introduction

2. Environment

3. Dependency

4. Implementation

4.1. Model Description

A schematic representation of the model is shown below.

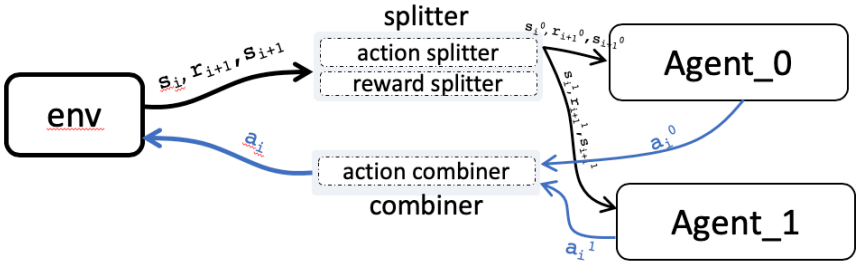


Fig. <>. A schematic representation of how the model is put together.

The model is not actually realized within a class. However, the astute observer will be able to recognize the rough structure within the training section in the file `src/utils/trainer.py`. It comprises of two agents that interact with the environment through a splitter/combiner. This splits the states (s_i), rewards (r_i), and *donnes* (not shown in the figure above), and combines the actions (a_i), from the agents, so that the agents are able to react to the environment.

Thus, the multi-agent problem is just broken down into two agents working independently. These will learn to react to the environment by themselves. Details of the agent and each of its components will be described in the following sections.

4.1.1. The Agent

A schematic diagram of the agent is shown in the figure below.

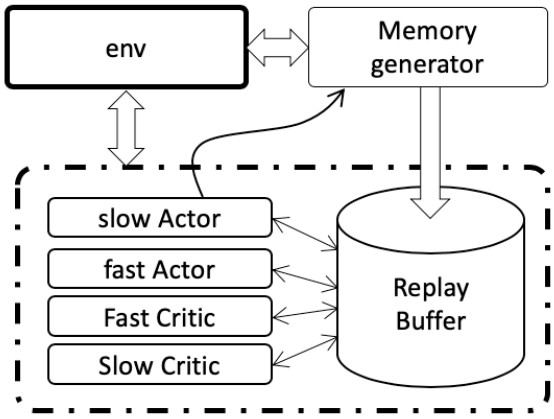


Fig. <>. A schematic diagram of the agent used by the model.

The agent comprises of two actors and two critics. It is intended to solve for the equation for RL:

$$q_i(s_i, a_i) = r_i + \sum_{j=i+1}^{\infty} \gamma^{j-i} r_j \tag{1}$$

In the equation above, s_i and a_i refer to the i^{th} state and action respectively, and γ represents the discount factor. Of course, we don't know the future reward. Hence it is typically approximated with what the future cumulative value *might have been*. This is typically rewritten in the following manner:

$$q_i(s_i, a_i) = r_i + \gamma q_{i+1}(s_{i+1}, a_{i+1}) \tag{2}$$

Note that not only do we not know the function q, we also have no idea how to come up with the action a. To solve this problem, we shall generate two *actors* and two *critics*. The actors take a state and predict an action, while a critic takes a state and an action and try to predict the Q value. We shall represent the actors by A^s and A^f for the slow- and fast-acting actors respectively. Similarly, we shall denote the critics by C^s and C^f. Using these, we shall rewrite eq. (2) by replacing the critics with the Q values.

$$C^f(s_i, a_i) = (1-\alpha)C^f(s_i, a_i) + \alpha[r_i + \gamma C^s(s_{i+1}, a_{i+1})] \tag{3}$$

Remember that this equation is not strictly a mathematical equation. It represents the update rule of the weights of the fast critic.

Now, let is replace the actors into eq. (3). This equation represents the learning equation for our fast critic. Notice that the slow critic is not changed. This is used for providing some stability to the critic for predicting the future rewards. This slow critic is periodically updated with the value of the fast critic with an τ that works in a manner similar to the learning rate α shown in eq. (3).

$$C^f\left(s_i, A^f(s_i)\right) = (1-\alpha)C^f\left(s_i, A^f(s_i)\right) + \alpha\left[r_i + \gamma C^f\left(s_{i+1}, A^s(s_{i+1})\right)\right] \tag{4}$$

Actors are just supposed to maximize the Q value. Hence the update equation is represented by the following equation:

$$A^f(s_i) = (1-\alpha)A^f(s_i) + \alpha C^s\left(s_i, A^s(s_i)\right) \tag{5}$$

Finally, the slow-moving actors and critics are periodically refreshed in a similar manner:

$$\begin{aligned} A^s &= (1-\tau)A^s + \tau A^s \\ C^s &= (1-\tau)C^s + \tau C^s \end{aligned} \tag{6}$$

The entire set of self-consistent equations are then recursively solved by going through interaction with the environment over an extended period of time until some solution criterion is reached. In this specific case, each actor has to be able to consistently hit the ball across 9 to 10 times.

4.1.1. 1. The Actors

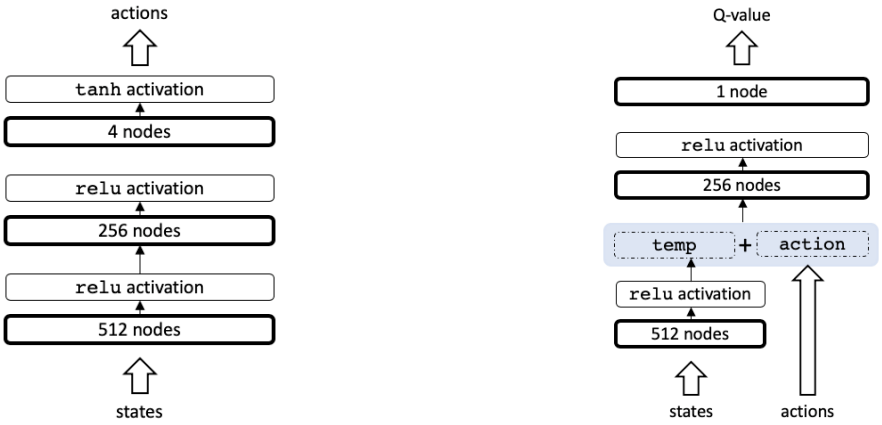


Fig. <>. A schematic representation of the actor (a) and the critic (b) is shown above.

The agent is a simple sequential neural network model that takes the state as input and generates the actions as output. It comprises of three layers with 512, 256 and 4 nodes each, with the first two layers having `relu` activation, while the final layer having the `tanh` activation. The `tanh` activation of the last layer allows the output to be bonded to somewhere between -1 and 1, as is the requirement of the four actions that the actor is supposed to return.

4.1.1.2. The Critics

A schematic representation of the critic is shown in Fig. <>. (b). The critic is also a fairly simple sequential neural network model. The state goes through a layer that has 512 nodes and is then activated by `relu` activation. This is then combined with the actor vector, and the result is then passed through the next layer containing 256 nodes with `relu` activation again. This is then passed onto the next layer that has a single node. This is the Q value. Note that the Q value is unbounded and linear, and thus is not activated.

4.1.1.3. The Replay Buffers

The replay buffers have been generated using a simple `deque`. The replay buffer is a class that has been defined within the file `src/utils/memory.py` by a class called `ReplayBuffer`. This exposes several methods that allow the replay buffer to be populated (`append`, and `appendMany`), sampled (`sample`) and also has methods that allow the `ReplayBuffer` to be saved (`save`) and reloaded (`load`) to and from disk at any point. This is especially useful because it is possible that we have already had a great set of memories that we can use. It is typically a total waste to throw away the entire buffer at one go. Hence, these buffers may be saved, so that they may be used in the future if needed.

There are several characteristics of the specific part of the game that makes the design of the memory buffer efficient.

1. The game is episodic.
2. Points are awarded every time the racquet of an agent is able to hit the ball across the net.
3. Each hit is fairly independent of the other hits. Hence the probability that a particular agent is going to be able to hit the ball across the net depends mostly upon the set of actions that the agent takes a few time-points (let's say 5 time points) before actually hitting the ball, at which point the racquet actually

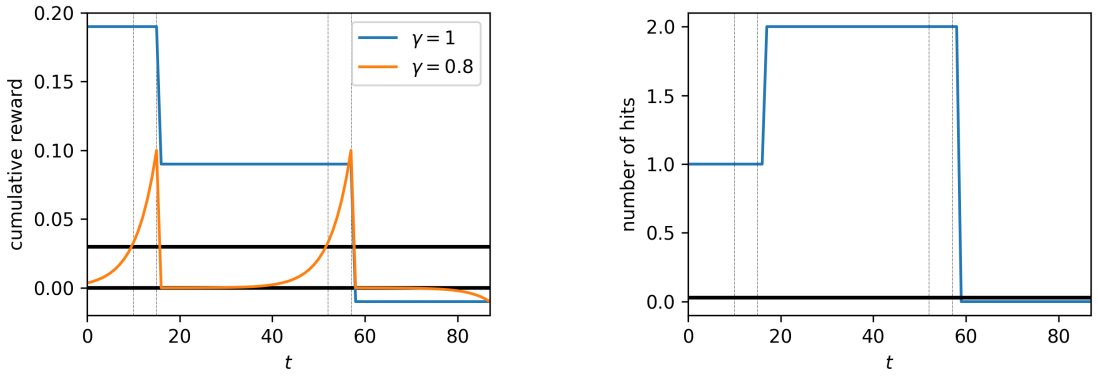
lobs the ball over the fence. Hence, it would be most ideal if the agent learns around this segment, rather than during all segments that the agent does not actually make a dent at the learning experiment.

4. Note that it is quite easily possible to calculate the actual cumulative reward for this problem for a fairly large number of episodes.

Hence, in this specific case, the memory buffer contains a tuple of the following form:

`(state, action, reward, next_state, done, cumRewards, numHits)`

The cumulative rewards (represented by `cumRewards`) are calculated with a γ of 1. However, the choice of the value of γ that should be used will be discussed in the next section. `numHits` represents the number of hits that a particular agent is having for a particular hit. This is easier to visualize with the following graphs



(a) (b)

Fig. <>. Different characteristics of an agent is shown for an entire episode. In this episode, the agent starts, and successfully lobs the ball over the net (approximately at a time point of 10 units), the other agent hits the ball back, and this agent is able to hit it one more time approximately around 60 time units. The cumulative rewards for the first agent is shown in (a) for two different values of γ, while the hit number is represented by the figure in (b).

Generating memories: Refreshing a memory buffer is a needed very often. Hence a function has been presented that will allow a set of two list of tuples to be generated. The function called `memories` is present in present in the file `src/utils/generateMemories.py`. This function and the ideas that went into designing this function will be briefly discussed in the following paragraphs.

Since we are able to calculate the cumulative reward for every point in an episode, this is exactly what has been done. An episode in played using a given `policy`[1] that will last at most N_{st} steps (or when a game stops).

Now, note that there is very little to be learnt from regions where the agent is unable to get a score. Hence, it would be great if we were able to learn from the region close to where the agent actually gets a point. For finding this reason, the cumulative rewards with a cumulative rewards is generated with a γ of 0.8 as shown in the Figure above. This shown that the cumulative rewards decreases exponentially, as we get farther away from the location of a successful hit. Here, we have found regions within the episode wherein this value of the cumulative reward remains above the value of 0.03. These are narrow regions demarcated by the thin gray lines in the Figures above. At least in the beginning, it is possible that it might be best to learn from within this region.

Sampling from memory: Although it might be possible to sample randomly form the memory buffer, a slightly smarter sampling methodology has been used. Sampling is performed based upon the cumulative rewards captured for each episode. The probability that a particular tuple i is sampled form the memory buffer is based upon the equation:

[1] Generation of policies for both generating memories and training is discussed in Section <Update this>.

$$p_i = \frac{x_i + \epsilon}{\sum_i [x_i + \epsilon]}$$

Here, x_i represents the i^{th} parameter and ϵ is a small number that determines how strict we want the input to be used for learning. Several different things have been experimented upon to get parameters that are most relevant for this process. We shall discuss the different attempts in the experimentation section.

Note that there are two reasons for adding the value ϵ. First, this will prevent any possibility of division by zero. Next, it will be possible to also include values for which the cumulative reward is very small. Tuning this parameter down (say to 0) will p_i be the same as that represented by the bare c_i. On the other hand, using a large value of say 1000 results in a all tuples being approximately equally likely to be sampled.

4.2. Training the Model

4.3. Experiments

5. Results

6. Summary

7. Acknowledgements

------- All this stuff needs to be deleted -----

```
maxScore for Agent 1 : 0.19000000320374966

rewards    1 :
[ 0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
  0.    0.    0.    0.    0.1   0.    0.    0.    0.    0.    0.    0.
  0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
  0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
  0.    0.    0.    0.    0.    0.    0.    0.    0.    0.1   0.
  0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
  0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
  0.    0.    0.    0.   -0.01]
Total Hits 1 : [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Cumrewards 1 :
[ 3.51873654e-03  4.39842067e-03  5.49802584e-03  6.87253230e-03
  8.59066538e-03  1.07383317e-02  1.34229147e-02  1.67786433e-02
  2.09733042e-02  2.62166302e-02  3.27707877e-02  4.09634847e-02
  5.12043558e-02  6.40054448e-02  8.00068060e-02  1.00008507e-01
  1.06325077e-05  1.32906347e-05  1.66132933e-05  2.07666166e-05
  2.59582708e-05  3.24478385e-05  4.05597981e-05  5.06997477e-05
  6.33746846e-05  7.92183557e-05  9.90229447e-05  1.23778681e-04
  1.54723351e-04  1.93404189e-04  2.41755236e-04  3.02194045e-04
  3.77742556e-04  4.72178195e-04  5.90222744e-04  7.37778430e-04
  9.22223038e-04  1.15277880e-03  1.44097350e-03  1.80121687e-03
  2.25152109e-03  2.81440136e-03  3.51800170e-03  4.39750212e-03
  5.49687766e-03  6.87109707e-03  8.58887134e-03  1.07360892e-02
  1.34201115e-02  1.67751393e-02  2.09689242e-02  2.62111552e-02
```

```
  3.27639440e-02  4.09549300e-02  5.11936625e-02  6.39920781e-02
  7.99900977e-02  9.99876221e-02 -1.54742501e-05 -1.93428127e-05
 -2.41785159e-05 -3.02231448e-05 -3.77789310e-05 -4.72236638e-05
 -5.90295797e-05 -7.37869746e-05 -9.22337183e-05 -1.15292148e-04
 -1.44115185e-04 -1.80143981e-04 -2.25179976e-04 -2.81474970e-04
 -3.51843713e-04 -4.39804641e-04 -5.49755802e-04 -6.87194752e-04
 -8.58993440e-04 -1.07374180e-03 -1.34217725e-03 -1.67772156e-03
 -2.09715195e-03 -2.62143994e-03 -3.27679993e-03 -4.09599991e-03
 -5.11999989e-03 -6.39999986e-03 -7.99999982e-03 -9.99999978e-03]
Cumreward1 1 : [ 0.19  0.19  0.19  0.19  0.19  0.19  0.19  0.19  0.19  0.19
0.19  0.19
  0.19  0.19  0.19  0.19  0.09  0.09  0.09  0.09  0.09  0.09  0.09  0.09
  0.09  0.09  0.09  0.09  0.09  0.09  0.09  0.09  0.09  0.09  0.09  0.09
  0.09  0.09  0.09  0.09  0.09  0.09  0.09  0.09  0.09  0.09  0.09  0.09
  0.09  0.09  0.09  0.09  0.09  0.09  0.09  0.09  0.09  0.09 -0.01 -0.01
 -0.01 -0.01 -0.01 -0.01 -0.01 -0.01 -0.01 -0.01 -0.01 -0.01 -0.01 -0.01
 -0.01 -0.01 -0.01 -0.01 -0.01 -0.01 -0.01 -0.01 -0.01 -0.01 -0.01 -0.01
 -0.01 -0.01 -0.01 -0.01]
```