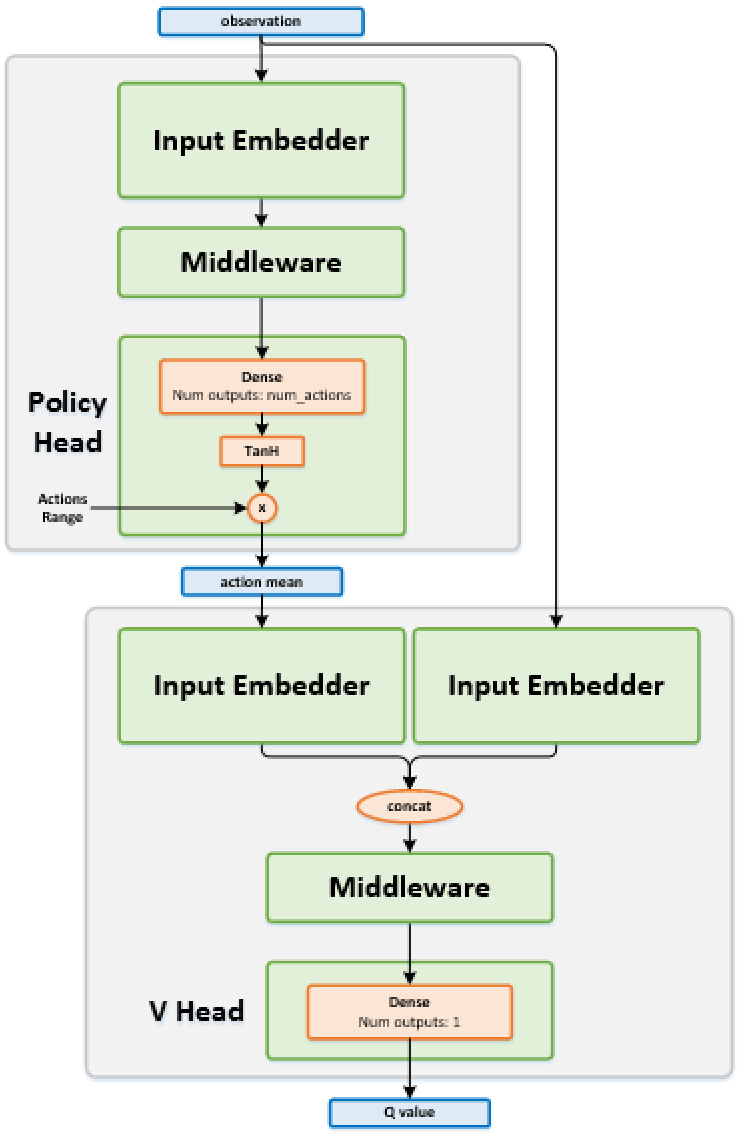
Report of Continues control

1. Define the Network:

Structure: MA-DDPG
I use 2 DDPG Agent, Delayed Deep Deterministic policy gradient in this practice.
And share their "*predict actions next*", and "*predict actions*"



Graph1, structure of DDPG. [2]

Network size:
As the Vector Observation space size in this environment is 24 each, action space size is 2 each.

I have defined First hidden layer units as 256 and 2nd hidden layer as 128 for Actor, small enough to run in CPU with the balance of simplicity and efficiency.

For the Critic First hidden layer units as 256 and 2nd hidden layer as 128,

2. Define the Replay Buffer:

Since the 2 agents observed different status and take different actions, so the reply buffer must be separated, so I linked 2 reply buffers for each.

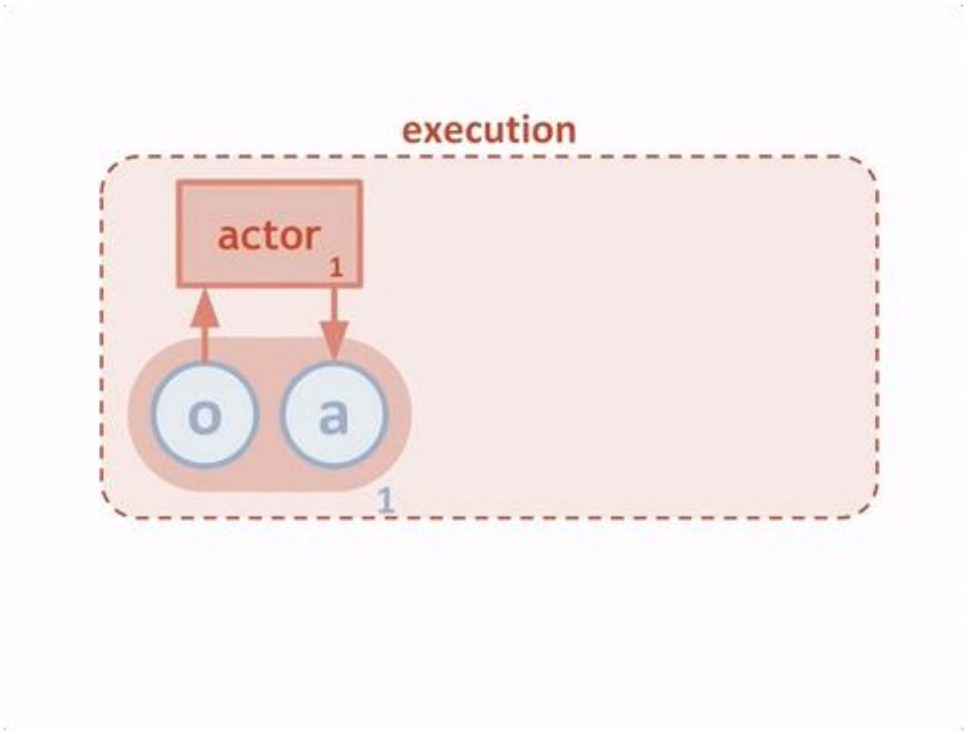
And I used a prioritized reply buffer, but not sum tree, takes N time rather than Log N to get random selections with prioritize and distribution. Hence, the size of the buffer should not be too long, so I take 1e5.

In each step of the episodes, push the full states, concoction of status of both agents, full action, full next status, individual reward and done to the buffer.

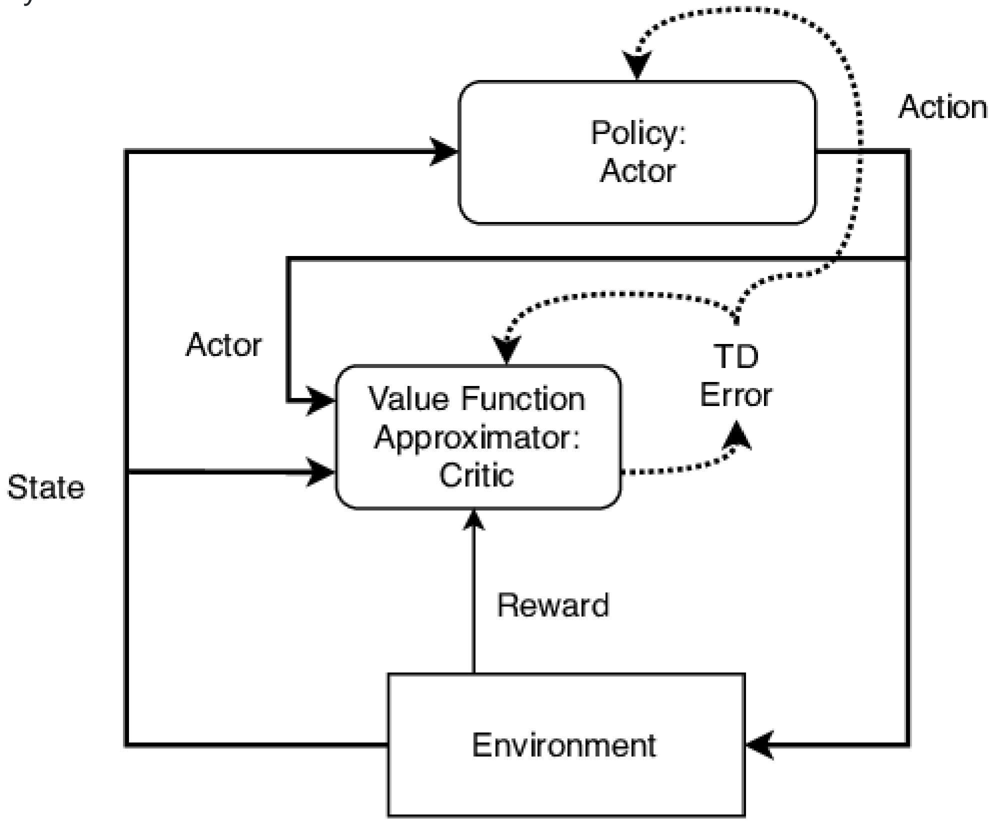
3. Define the Agent:

2 Levels of the Agent, DDPG agent, focus on basic DDPG algorithm, and a MADDPG agent which will coordinate data share between 2 DDPG agent.

Each Agent takes full status and full actions as input, actors will generate the Individual actions, and concatenate the generated action of the other agent, as input to the critic.



Each DDPG has 2 set of Networks as always: local and target, the local network will used for generating actions, and the target network will be used as updating the policy



4. Training the MADDPG:

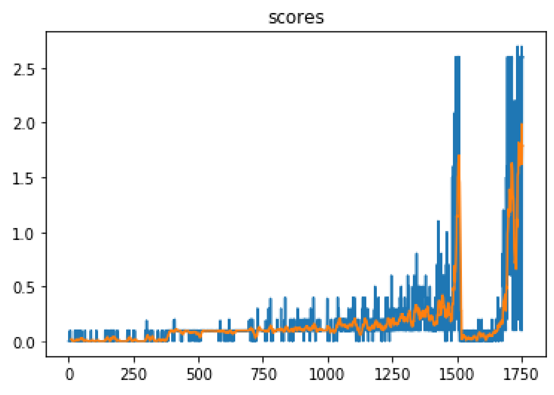
- Due to the Network is bigger than 2 projects before, and share middle level info, requires additional feed forward calculation. Furthermore, priority reply buffer needs one more feed forward, and random sample with priority distribution. So the training takes much more time than previous projects.

- Max_Steps: 1001 as observed in the env, it will be finished in 1001
- Skip_timesteps, 100, since very rare hit ball in the very begging.
- Random action generation, random normal distribution
- Batch_size, 128 is sufficient , 256 would be too slow.

```
Episode 100    Average Score: 0.010
Episode 200    Average Score: 0.013
Episode 300    Average Score: 0.008
Episode 400    Average Score: 0.033
Episode 500    Average Score: 0.086
Episode 600    Average Score: 0.090
Episode 700    Average Score: 0.092
Episode 800    Average Score: 0.097
Episode 900    Average Score: 0.108
Episode 1000   Average Score: 0.117
Episode 1100   Average Score: 0.136
Episode 1200   Average Score: 0.137
Episode 1300   Average Score: 0.174
Episode 1400   Average Score: 0.239
Episode 1500   Average Score: 0.352
Finished at Episode 1510     Reach Average Score: 0.506!
```

And after that, I have continued train the model to 1.0

```
Episode 100    Average Score: 0.038  Max Score: 0.20
Episode 200    Average Score: 0.373  Max Score: 2.60
Finished at Episode 246      Reach Average Score: 1.020!
```



5. Interesting observations:

For sharing the intermedia information among agents, I see someone has done alternatively [3], rather than sharing the predicted action and next actions with actor_target and actor_local, he arbitrarily shared the actual action of the other agent as the predict action and next action as input to the critic.

With the way, the calculation would definitely less, according to his training record, it converges faster, but I can't reproduce it, I had try his method, with switch "**SHARE_ACTUAL_ACTION**" in my code, seems very hard to converge at score around 0.3.

6. To do next:

1) Adopt TD3 in to MADDPG
2) Implement priority reply buffer with sumtree, saves cost from N to Log N
3) Train the score to >3.0

7. Reference:

1. **"Addressing Function Approximation Error in Actor-Critic Methods"** https://arxiv.org/abs/1706.02275
2. DDPG Network structure, https://nervanasystems.github.io/coach/components/agents/policy_optimization/ddpg.html
3. PHRABAL implementation https://github.com/PHRABAL/Tennis-MADDPG-PER/blob/master/MADDPG_PER.ipynb
4. Gtg's implementation https://github.com/gtg162y/DRLND/blob/master/P3_Collab_Compete/Tennis_Udacity_Workspace.ipynb