Basic concept

1)Multi-agent system
Multi-agent can share experiences and is robust

2)Multi-agent reinforce learning(MARL)
it is assumed that the environment is stationary(single-agent).
The matter agent approach : each agent have different observation of the environment state.
Mix cooperative and competitive behavior
maximize reward

3)multi-agent DDPG
the normal agents are rewarded based on the least distance of any of the agents to the landmark.
Penalized based on the distance between adversary and the target landmark.

4)monte carlo tree search

Boardgame: s_t Player: $(-1)^t$

Player $+1$:

- Performs actions a_{2t}
- Goal: maximize the score z

Player -1:

- Performs actions a_{2t+1}
- Goal: maximize the score $-z$

One common policy:

$$\pi_\theta\big(a_t|(-1)^t s_t\big)$$

One common critic:

$$\text{Estimates } (-1)^t z \longrightarrow v_\theta\big((-1)^t s_t\big)$$

random sampling

N = visit count

V = expected score

$$U = V + \frac{\sqrt{N_{\text{tot}}}}{1+N}$$

Exploitation Exploration

choose highest N, U for select branch
expansion ,back-propagation: update the statistic on the previous node,N→ V→ U
choose action by max U

MCTS Summary

Initialize top-node for current state, loop over actions for some N_{tot}:

1. Start from the top-node, repeatedly pick the child-node with the largest U

2. If $N = 0$ for the node, play a random game.
   Else, expand node, play a random game from a randomly selected child

3. Update statistics, back-propagate and update N and U as needed

Select move with highest visit counts

5)guide tree search

Exploration guided by the Policy $\pi_\theta\left(a_t|(-1)^t s_t\right)$ = Expert Policy

Simulation done by the Critic $v_\theta\left((-1)s_t\right)^t$ = Expert Critic

$\pi_\theta\left(a_t|(-1)^t s_t\right)$ = Expert Policy

$v_\theta\left((-1)s_t\right)^t$ = Expert Critic

N = visit count

V = expected score

$$U = \underbrace{V}_{\text{Critic controlled exploitation}} + \underbrace{c}_{\text{Hyperparameter}}\,\underbrace{\pi_\theta\left(a_t|(-1)^t s_t\right)\frac{\sqrt{N_{\text{tot}}}}{1+N}}_{\text{Policy guided exploration}}$$

policy focus on exploring moves that an expert is likely to play.

Critic estimate the outcome of a game without running a simulation

4)self-play training

Action probability:

$$p_a^{(t)} = \frac{N_a^{(t)}}{\sum_a N_a^{(t)}}$$

loss function to closer result of monte carlo tree search

Improves critic

$$L(\theta) = \sum_t \left\{ \left[v_\theta\big((-1)^t s_t\big) - (-1)^t z \right]^2 - \sum_a p_a^{(t)} \log \pi_\theta\big((-1)^t s_t\big) \right\}$$

Improves Policy

5)alphazero

AlphaZero Algorithm

1. Initialize network for critic and policy (v_θ, π_θ)
2. Play a game using MCTS
3. Compute $L(\theta) = \sum_t \left\{ \left[v_\theta\big((-1)^t s_t\big) - (-1)^t z \right]^2 - \sum_a p_a^{(t)} \log \pi_\theta\big((-1)^t s_t\big) \right\}$, perform gradient descent
4. Repeat Step 2-3

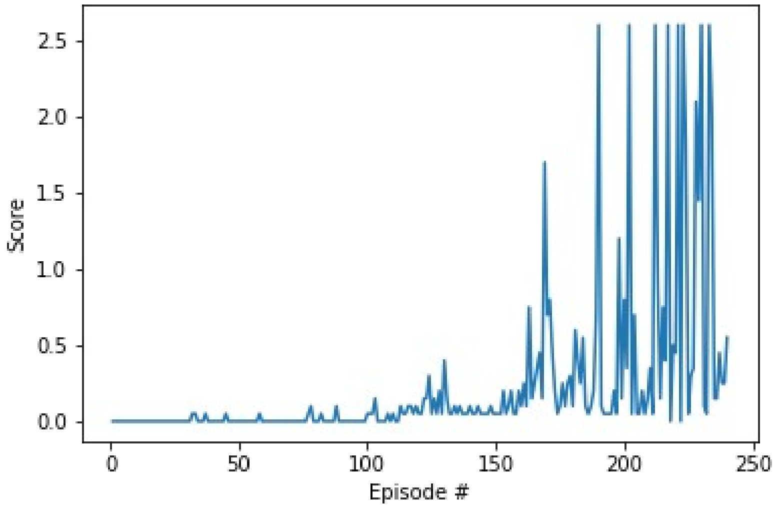
code(reference to Tomas0413/Collaboration-and-Competition)

```
def step(self, state, action, reward, next_state, done, current_time_step):
      """Save experience in replay memory, and use random sample from buffer to learn."""
      # Save experience / reward
      self.memory.add(state, action, reward, next_state, done)

      # Learn, if enough samples are available in memory
      if len(self.memory) > BATCH_SIZE and current_time_step % 
LEARN_EVERY_X_TIMESTEPS ==0:
          for update_step_num in range(UPDATES_PER_LEARN_STEP):
              experiences = self.memory.sample()
              self.learn(experiences, GAMMA)

def act(self, state, add_noise=True):
      """Returns actions for given state as per current policy."""
      state = torch.from_numpy(state).float().to(device)
      self.actor_local.eval()
      with torch.no_grad():
          action = self.actor_local(state).cpu().data.numpy()
```

```
    #print(action)
self.actor_local.train()
if add_noise:
    for i  in range(LEARN_EVERY_X_TIMESTEPS):
        action[i] += self.epsilon * self.noise.sample()
return np.clip(action, -1, 1)
```



Parameters

LEARN_EVERY_X_TIMESTEPS = 2 # agent number
BUFFER_SIZE = int(1e5) # replay buffer size
BATCH_SIZE = 128   # minibatch size
GAMMA = 0.99   # discount factor
TAU = 1e-3   # for soft update of target parameters
LR_ACTOR = 1e-4   # learning rate of the actor
LR_CRITIC = 1e-4   # learning rate of the critic
WEIGHT_DECAY = 0   # L2 weight decay

UPDATES_PER_LEARN_STEP = 10
EPSILON = 1.0
EPSILON_DECAY = 1e-6

DDPG layer size
actor : 256→ 128
critic : 256→128

Future

different task may need different type agent