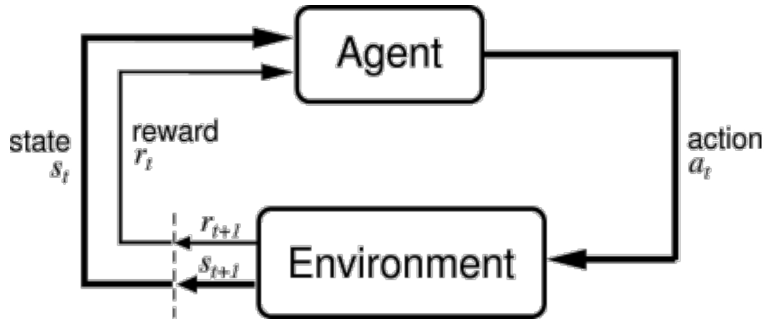
Multi Agent Deep Reinforcement Learning

By: Karkala Shashank Hegde
Guided by: Professor Keith Chugg

What is Multi agent RL?

Markov Decision Process

- Consider a system which defines the interaction between a agent and its surrounding environment.
- The environment represents the state at time t as a vector s_t.
- The agent's actions at time t can be represented as a_t.
- Reward is provided as a feedback by the environment as r_t.
- This can be iterated to get values at next time step t+1.
- We also have a boolean value attached to the feedback, this denotes if the environment has reached a terminal state.



Mathematical Representation of agent and Environment

Agent can be considered a function of the state, and predicts the actions (or sometimes the probabilities of actions).

ie, `π(s) → a`

Sometimes, the policy can be *stochastic* instead of *deterministic*. In such a case, instead of returning a unique action `a`, the policy returns a probability distribution over a set of actions. But for this presentation we shall only consider the deterministic case.

The environment can be thought of two functions:

- Reward Function: $$\mathcal{R}_s^a = \mathbb{E}\left[R_{t+1} \mid S_t = s, A_t = a\right]$$
- Transition Probabilities: $$\mathcal{P}_{ss'}^a = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$$

<u>These functions are non stochastic. I.e., Reward for a state-action today is the same as tomorrow</u>

Note the markov property of the the functions (The next value of the function only depends on the current value)

Goal

Maximize immediate reward, r_t? NO. Usually it would be beneficial to sacrifice immediate rewards if it would result in increased future rewards.

Thus we need “strategies” and not just search of actions that lead to maximum immediate reward.

Therefore we calculate the Q (for quality) value. This is the infinite sum of future discounted rewards if we used the current policy π. Let gamma γ be the discount factor

$$Q^{\pi}(s,a) = E_{\pi}\{\sum_{k=0}^{\infty} \gamma^{k} r_{t+k+1} | s_t = s, a_t = a\}$$

Therefore we need to adjust our policy to that maximises this Q value for every state action pair. I.e, our policy needs to take actions that maximise the Q value.

Note

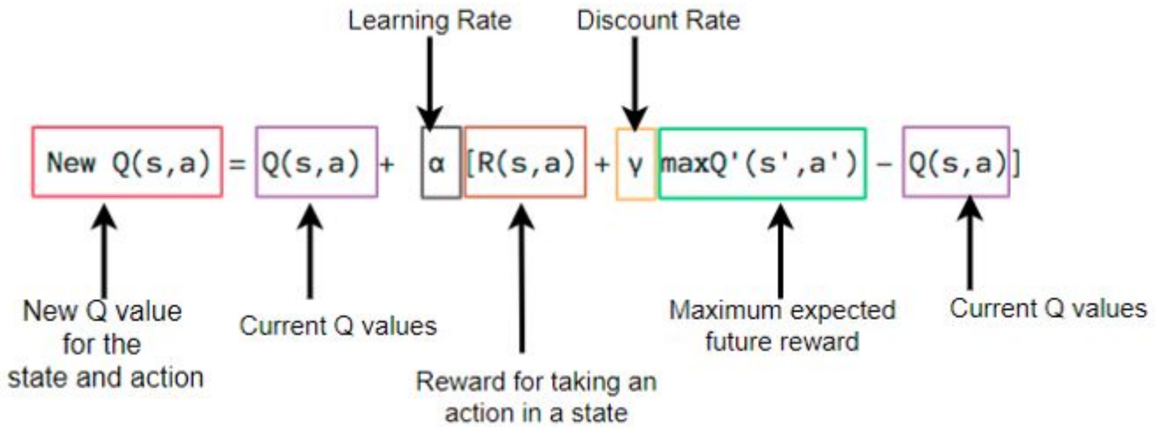
- There are many other ways to get optimal policies, these include PPO, TRPO, IMPALA, Imitation learning, AlphaGo etc.
- For this presentation we shall only look at methods that optimize policies over Q values

https://arxiv.org/pdf/1708.05866.pdf

Two part problem

1. Estimate Q value
2. Adjust Policy to maximize the estimated Q value

How to estimate Q value?
Bellman equation (Fight On)!


Learning Rate
Discount Rate
New Q(s,a) = Q(s,a) + α [R(s,a) + γ maxQ'(s',a') − Q(s,a)]
New Q value for the state and action
Current Q values
Reward for taking an action in a state
Maximum expected future reward
Current Q values

https://towardsdatascience.com/reinforcement-learning-markov-decision-process-part-2-96837c936ec3

Deep Q Networks (DQN)

- Use a Neural Network to estimate the Q value.
- For a discrete action space (eg. Up, Down, Left, Right), **a greedy policy can be choosing an action that gives the maximum predicted Q value for a given state**.
- Explore randomly sometimes to get unseen state-action pairs and their corresponding rewards.
- sample a batch of state, action and reward batch.
- For a given set of state and action pair, calculate target Q value using Bellman eqn.
- Use Mean Squared error to tune neural network.

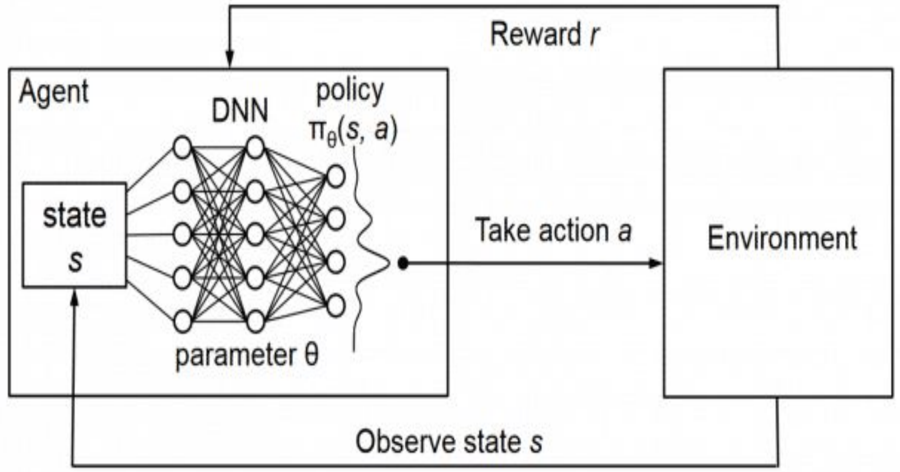
https://arxiv.org/abs/1312.5602

DQN Limitation

- Cannot be used for Continuous action space.
- Since we use a greedy policy to select action that gives maximum Q value, the actions have to be countably finite.
- A large action dimension implies a large Q network, and we'll have to do many forward passes to estimate Q for each action, just to select the next step

Solution for Continuous Action Space

- Use a Neural Network to model the policy π as well.
- Since the output layer can be Linear layer of any dimension, they can predict any N dimension action



Deep Deterministic Policy Gradients (DDPG)

Two Neural Networks:

- Q Network (Critic):

  Estimates the Q value for a given state, action pair.

  Calculate target using Bellman Eqn, update using MSE loss
- Policy Network (Actor):

  Predicts the continuous action for a given state.

  Directly use the negative Q value for the predicted action and given state as the loss.

https://arxiv.org/pdf/1509.02971.pdf

Multi Agent Reinforcement Learning

A simple approach would be using multiple RL agents each maintaining its own actor critic pairs, each agent learns independently.

Problems:

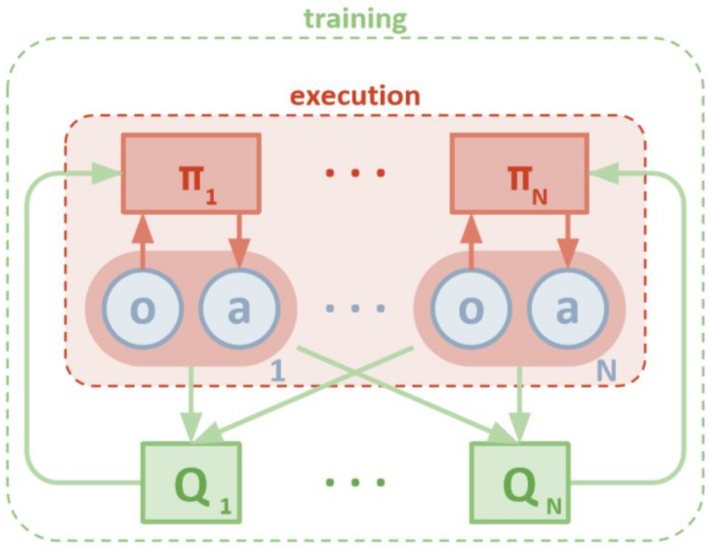
- An agent should not have access to other agents states during execution
- For one agent, it’s surroundings forms the environment, therefore it’s environment would include all other agents. If these agents keep learning, the environment functions become stochastic.
- How to promote competition or cooperation between agents?

Multi Agent Deep Deterministic Policy Gradients (MADDPG)

Similar to the simple approach, each agent maintains a policy network and a Q network.

During execution, each agent's policy uses its own state space while predicting actions.

During training, for each agent, while estimating Q value, **we concatenate every agent's state and actions.**

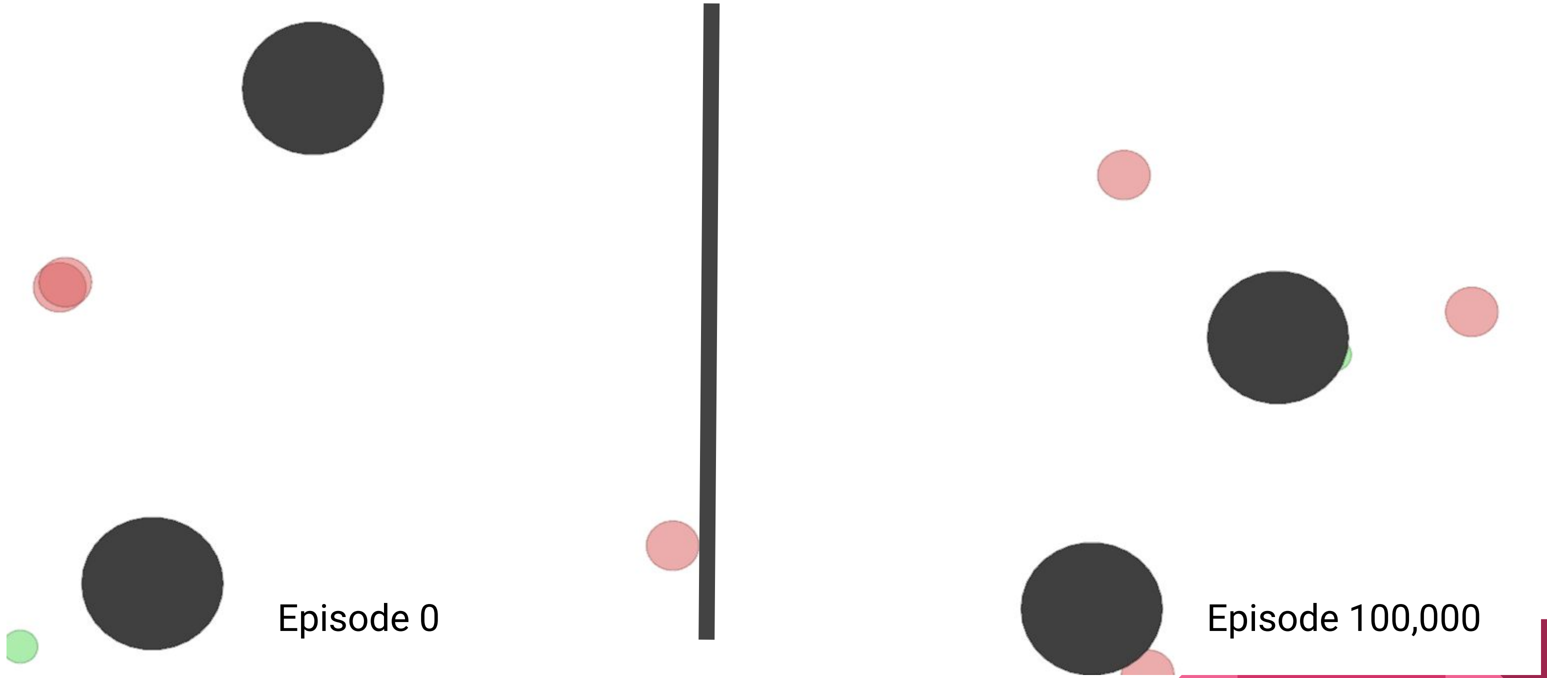


https://arxiv.org/pdf/1706.02275.pdf

MADDPG (cont)

- Centralized training with decentralized execution
- Since each agent's Q network takes every agent's concatenated state and concatenated action, we can make the reward function non stochastic.
- Also, since each agent receives its own reward, it can form policies to be competitive against or cooperative with other agents.
- Finally, during execution, ignore the Q network, and just use the policy network to predict the action.

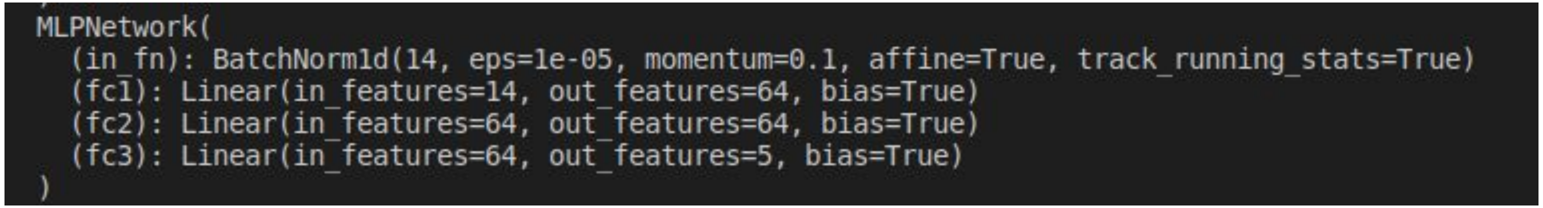
Demo: Predator-Prey



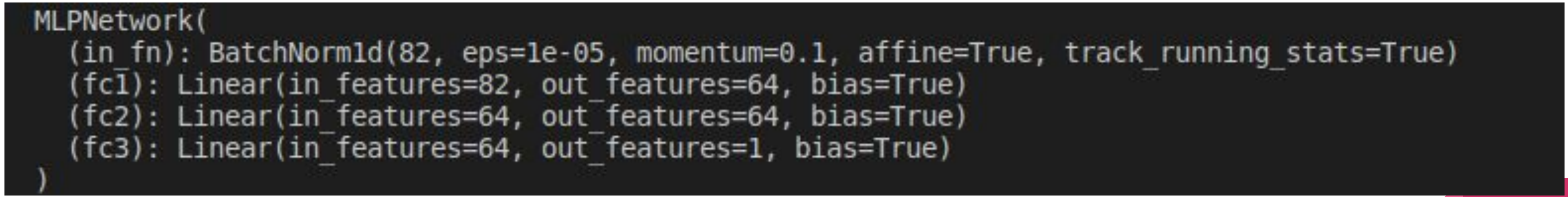
Gym environment: https://github.com/openai/multiagent-particle-envs

Models

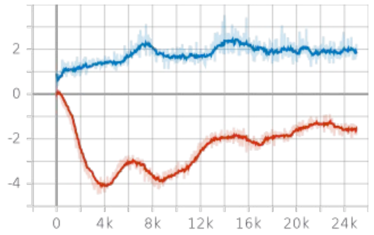
Policy


```
MLPNetwork(
  (in_fn): BatchNorm1d(14, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (fc1): Linear(in_features=14, out_features=64, bias=True)
  (fc2): Linear(in_features=64, out_features=64, bias=True)
  (fc3): Linear(in_features=64, out_features=5, bias=True)
)
```

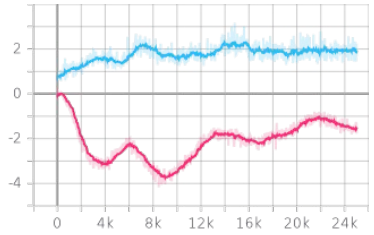
Q network


```
MLPNetwork(
  (in_fn): BatchNorm1d(82, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (fc1): Linear(in_features=82, out_features=64, bias=True)
  (fc2): Linear(in_features=64, out_features=64, bias=True)
  (fc3): Linear(in_features=64, out_features=1, bias=True)
)
```

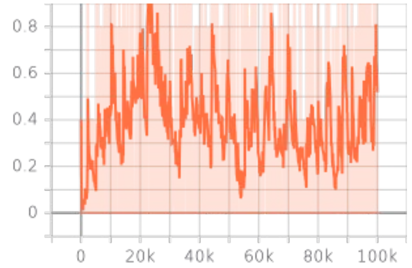
Learning Curves



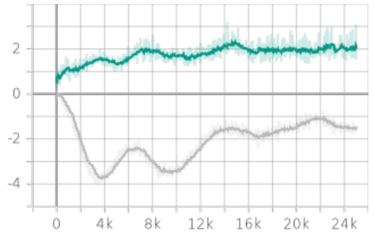
agent0



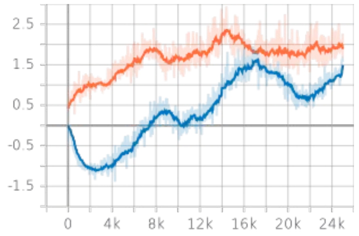
agent1



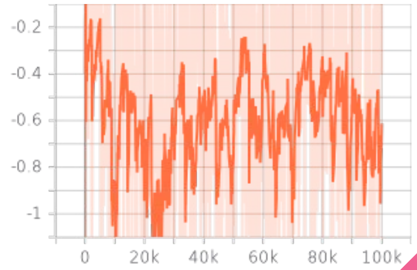
Mean Episode reward



agent2



agent3



Thank you