REPORT for Project 3: Collaboration and Competition

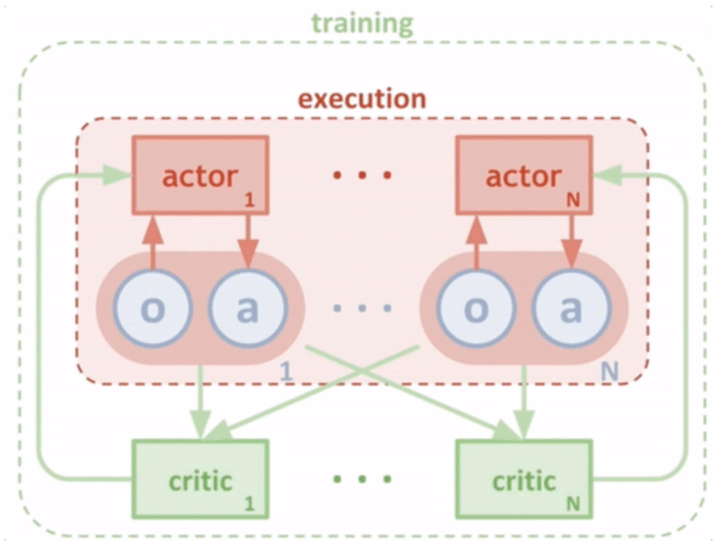
I implemented the DDPG algorithm for multi-agent environment and achieved the required score in Trained in 1085 episodes of training.

THE ALGORITHM:

DEEP DETERMINISTIC POLICY GRADIENTS FOR MULTI-AGENT ENVIRONMENT

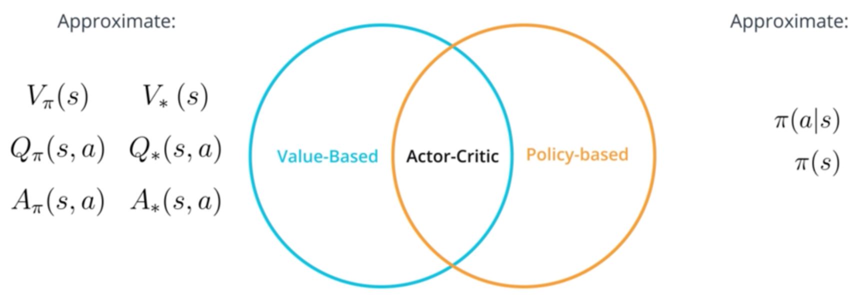
Based on my understanding of the multi-agent systems; we could approach to this problem in the following ways;

- By creating **independent** actor and critic networks for each agent in the environment and training them with interaction and self-play. Each agent see the other agent as part of the environment. This is simpler approach and similar to real-life learning.
- By creating the **shared** actor or/and policy networks, and even shared memory .
  This approach can also be referred as 'Centralized training – Decentralized execution' (individual actor networks and shared critic network, or 'Decentralized training – o Centralized execution -individual actor networks and shared critic network
  This approach requires more complex implementation but leads to faster convergence as agents sharing their data and resources.



As there is no specific requirement for this project; I used first approach and train the agents independently. Each agent see the other agent as part of the environment and based on its local observation it learns the best parameters to maximize the score.

While using the above approach to solve this Multi-agent environment; I used the DDPG (Deep Deterministic Policy Gradients) algorithm to solve the given environment. DDPG is an actor-critic algorithm where combine both value based and policy based algorithms.

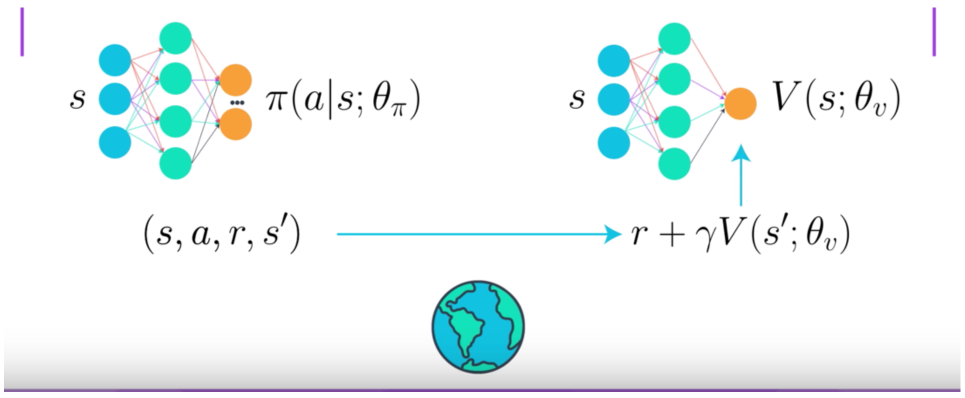


In actor critic methods; there are 2 neural networks, one for policy-based calculations (the Actor Network) and one for value-based calculations. (the Critic Network) . The actor uses the neural network to estimate the successful action and the critic network uses the neural network to estimate the value-function of the states.

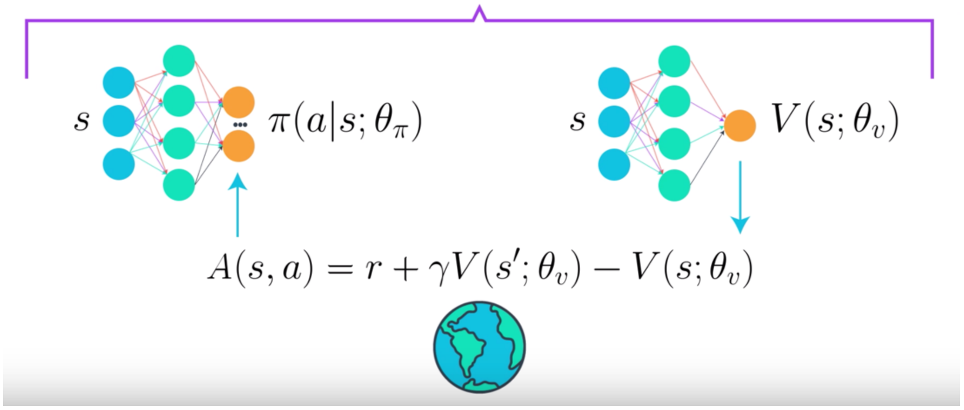
As our problem requires 2 agents, I created 2 agent objects which has its own actor and policy networks.

Basically after initialization of networks; each agent’s actor network estimates an action based on the initialized policy based on initial parameters (it also update itself based on initial version of critic network) and send it to environment then we observe the next state and reward from environment.

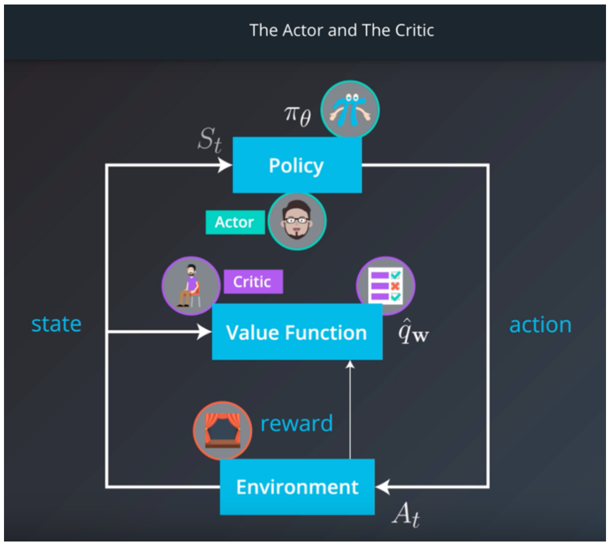
Then we store s,a,r,s’ separately for each agent at their memory objects. Then send the state vector to actor network and state-action pair to critic network.

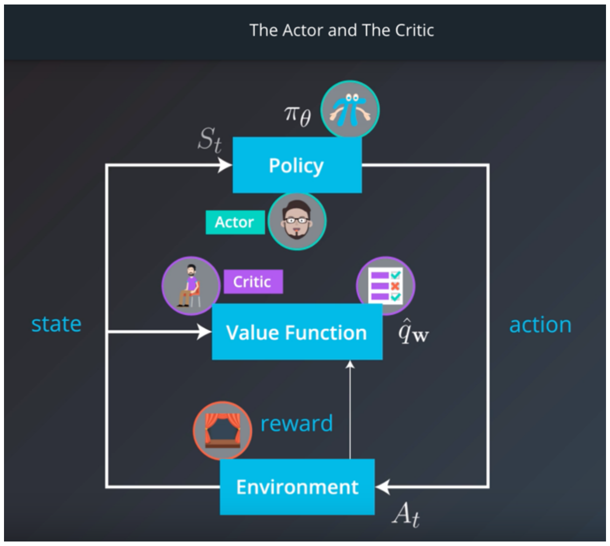


Critic network estimates the q-value for the given state-action pair for its agent and feed the agent's actor's loss calculation for its update.



At below you can see the overall DDPG process in one chart. This works for both agents parallelly.





HYPERPARAMETERS:

In order to achieve required score, hyperparameters are optimized as below;

```
BUFFER_SIZE = int(1e5)  # replay buffer size
BATCH_SIZE = 64       # minibatch size
GAMMA = 0.98           # discount factor
TAU = 1e-2            # 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
THETA=0.6, SIGMA=0.2.   # OUnoise
```

I implemented same hyper parameters from the previous project and only changed discount and ounoise hypermaters.

- The same hyperparameters from DDPG paper couldn't be used as they weren't able to solve the environment. Thus I decrease the TAU to 1e-2 as it creates more updated target network faster compared to original 1e-2.
- After many tests; instead of 1e-3 from original paper; I decided to use the learning rate 1e-4 for the critic network. It works better, probably it stabilizes the learning of critic network further, even though it leads to slower learning.
- Again after doing many tests; I changed the OUnoise parameters to theta=0.6, sigma=0.2 (while in original DDPG paper, they were 0.15 and 0.2) as those provided better performance, probably because of good mix of exploration and exploitation.

NN- MODEL

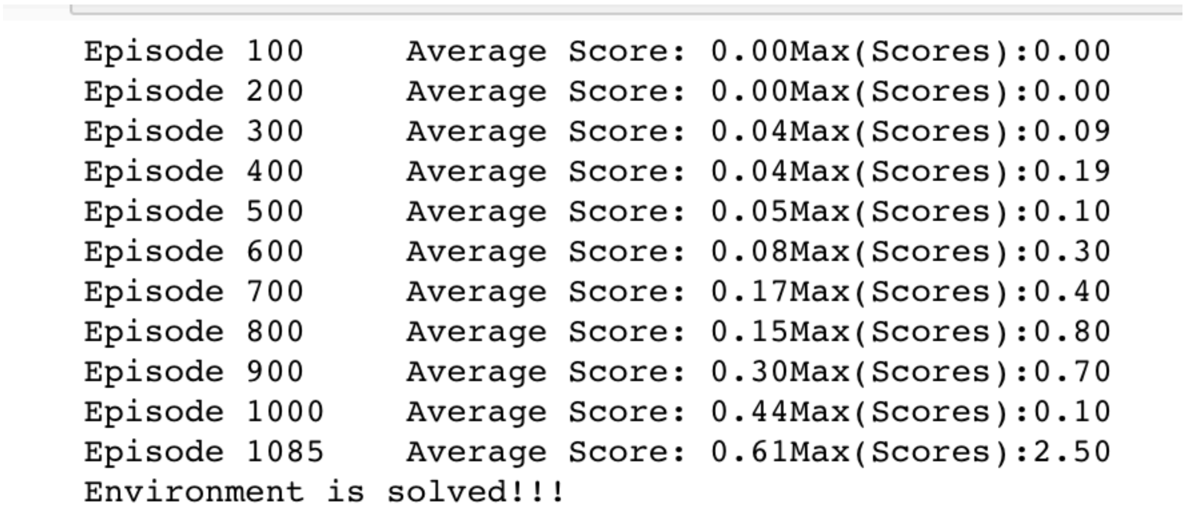
Regarding the neural architectures; I used exactly same model for both agents as the task for both are exactly same. I added 2 hidden layers to each actor and critic networks and used the 400&300 hidden units for actor and 400&400 hidden units for critic network as it leads to better training.

```
class Actor(nn.Module):
    """Actor (Policy) Model."""
        self.fc1 = nn.Linear(state_size, fc1_units)
        self.fc2 = nn.Linear(fc1_units, fc2_units)
        self.fc3 = nn.Linear(fc2_units, action_size)
        """Build an actor (policy) network that maps states -> actions."""
         xs = F.leaky_relu(self.fcs1(state))
        x = torch.cat((xs, action), dim=1)
        x = F.leaky_relu(self.fc2(x))
class Critic(nn.Module):
    """Critic (Value) Model."""
        self.fcs1 = nn.Linear(state_size, fcs1_units)
        self.fc2 = nn.Linear(fcs1_units+action_size, fc2_units)
        self.fc3 = nn.Linear(fc2_units, 1)
        """Build a critic (value) network that maps (state, action) pairs -> Q-values."""
        xs = F.leaky_relu(self.fcs1(state))
        x = torch.cat((xs, action), dim=1)
        x = F.leaky_relu(self.fc2(x))
        return self.fc3(x)
```

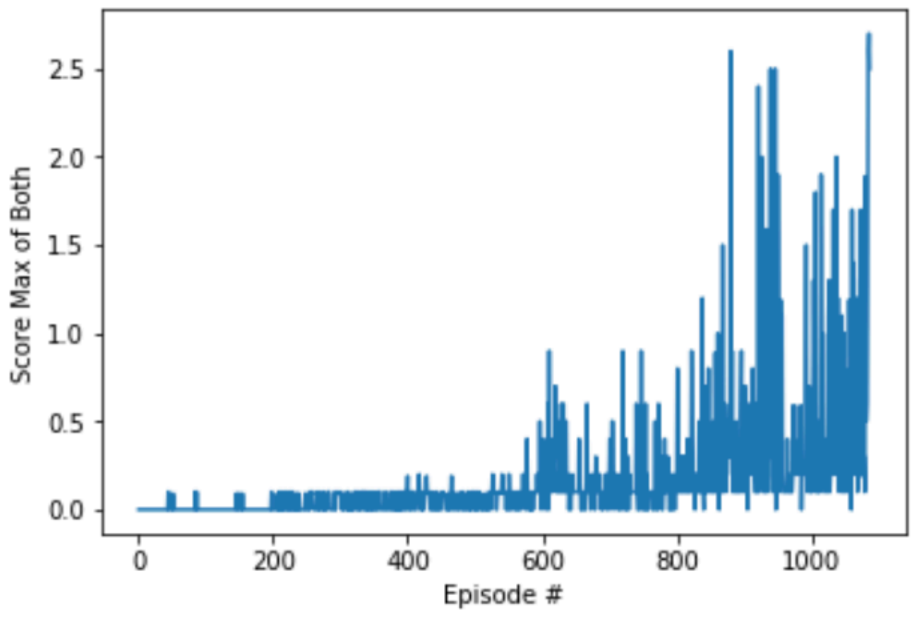
Additionally I used leaky relu for all all hidden layers (as suggested in paper) and finish the actor with tanh activation (due to action range +1,-1) for each agent and finish the critics with logits.

REWARDS:

Please see the plot of the rewards .



```
Episode 100     Average Score: 0.00Max(Scores):0.00
Episode 200     Average Score: 0.00Max(Scores):0.00
Episode 300     Average Score: 0.04Max(Scores):0.09
Episode 400     Average Score: 0.04Max(Scores):0.19
Episode 500     Average Score: 0.05Max(Scores):0.10
Episode 600     Average Score: 0.08Max(Scores):0.30
Episode 700     Average Score: 0.17Max(Scores):0.40
Episode 800     Average Score: 0.15Max(Scores):0.80
Episode 900     Average Score: 0.30Max(Scores):0.70
Episode 1000    Average Score: 0.44Max(Scores):0.10
Episode 1085    Average Score: 0.61Max(Scores):2.50
Environment is solved!!!
```



IDEAS for FUTURE WORK :

- First of all there can be better hyperparameters values, however I want to specifically test the different the neural architectures and activations.
- We can also try to use shared actor or/and critic networks for agents, similar to Open AI MADDPG paper.
- I think using parallel environments for training, similar to OPEN AI implementation can also give better results.

Progress Videos;

I also uploaded the training videos to the youtube. You can watch those at below links:

Without Training:
https://youtu.be/5B2y8rOncAE

Training :

https://youtu.be/gs503f_Kn48

Smart Agent:

https://youtu.be/Y93H6IBCpM0

References: All images and charts (except plot of rewards) are from Udacity lecture notes.