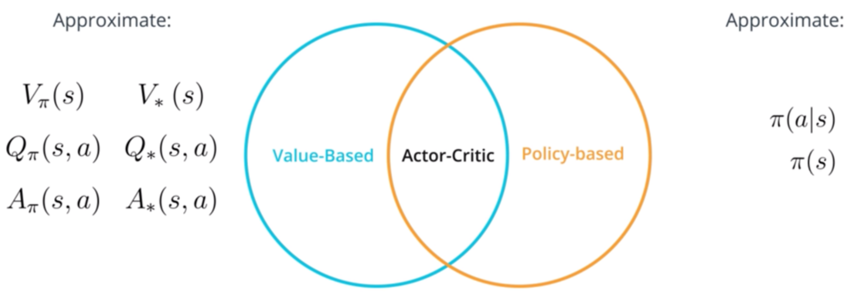
REPORT for Project 2: Continuous Learning

I implemented the mix of DDPG algorithm provided in the github repo. (For Bipedal and Pendulum) and achieved the required score in Trained in 213 episodes of training.

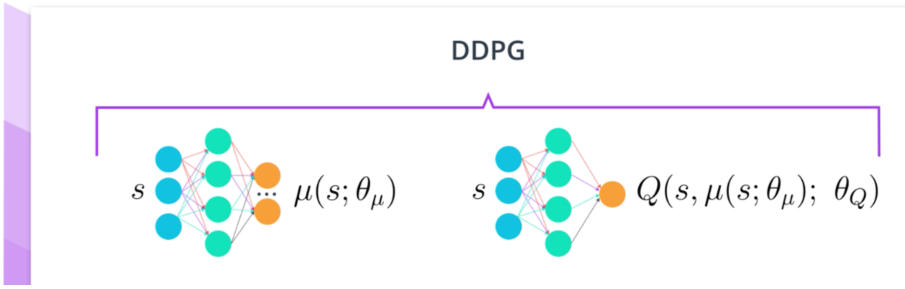
THE ALGORITHM

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.

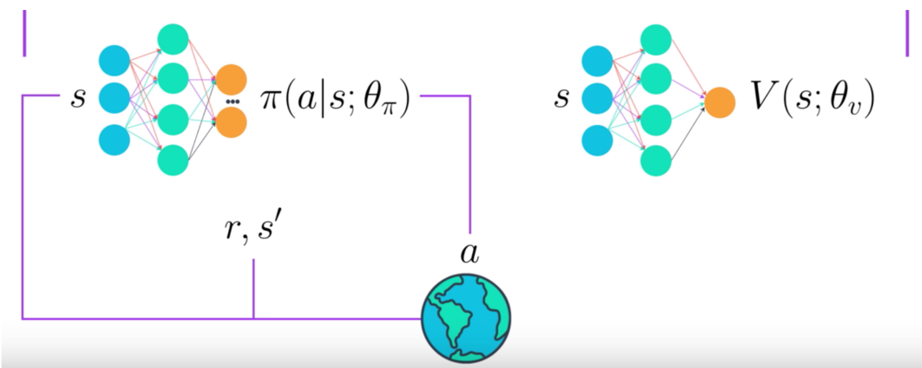


In actor critic methods; there are 2 neural networks, one for policy-based calculations (the Actor Network) and one for value-based calculations. (the Critic Network) . The actor uses the neural network to estimate the successful action and the critic network uses the neural network to estimate the value-function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

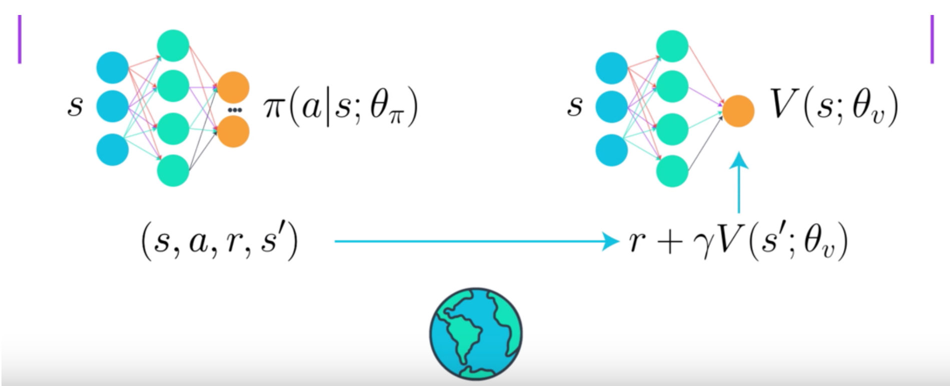
DDPG: Deep Deterministic Policy Gradient, Continuous Action-space



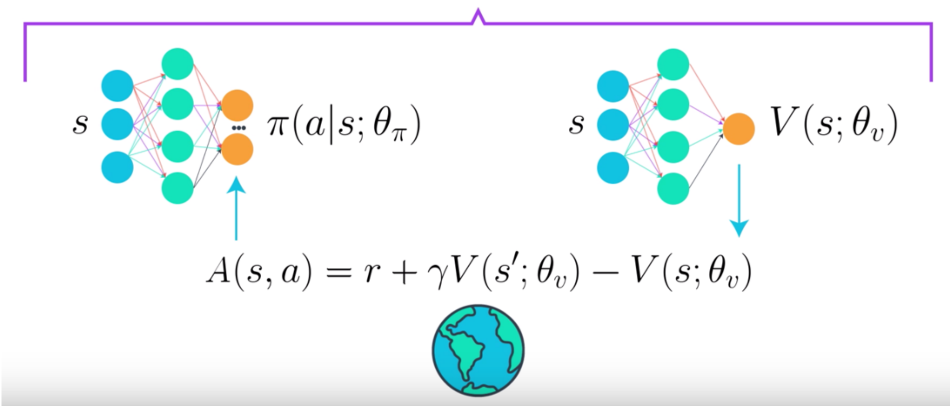
Basically after initialization of networks; actor network estimate an action based on the initialized policy based on initial parameters (it also update itself based on initial version of critic network) and send it to environment then we observe the next state and reward from environment.



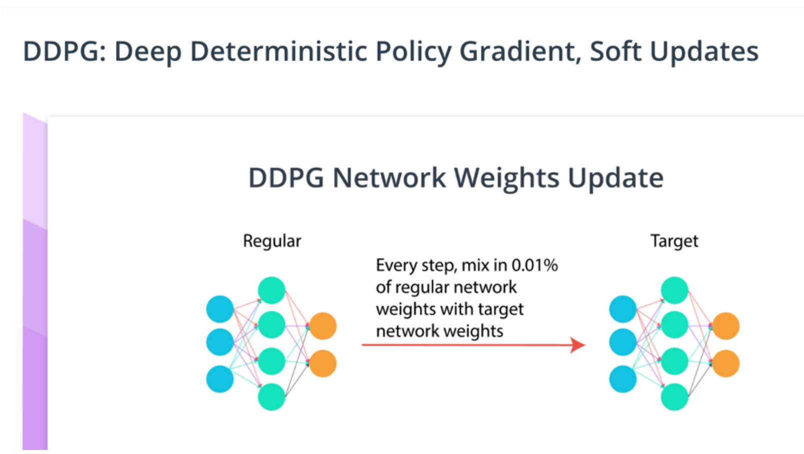
Then we store s,a,r,s’ tuple and send the state vector to actor network and state-action pair to critic network.



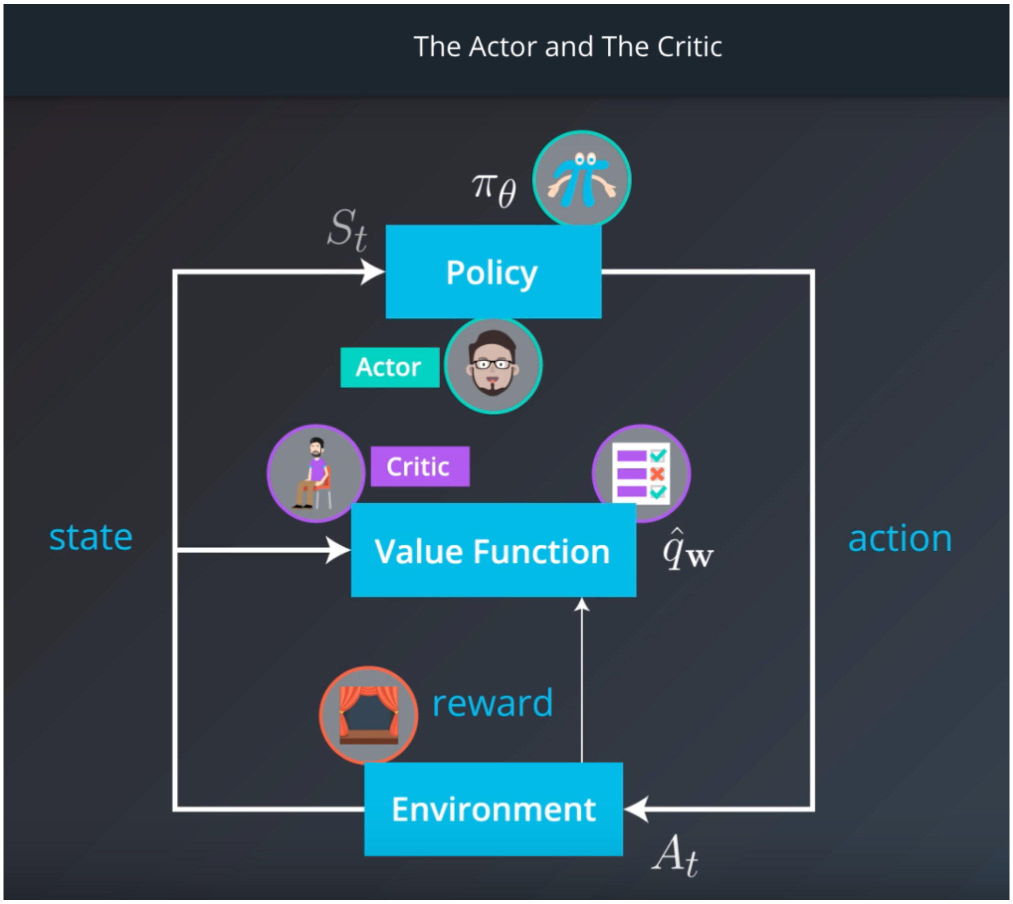
Critic network estimates the q-value for the given state-action pair and feed the actor's loss calculation for its update.



For backpropagation step (to calculate loss); both networks use the fixed (and softly updated in a coded way) copy of itself as a target network. And we want these updates to be soft to stabilize the learning.



At below you can see the overall DDPG process in one chart.



HYPERPARAMETERS:

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

BUFFER_SIZE = int(1e6) # replay buffer size

BATCH_SIZE = 64   # minibatch size

GAMMA = 0.99   # 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=0.5, sigma=0.3   # OUNoise

- 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=0.5, sigma=0.3 (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.

The other hyperparameters were the same as DDPG paper .

NN- MODEL

Regarding neural architecture; I used the mix of NN used in DPPG Pendulum and Bipedal code from class. It is 2 hidden layers with 400 and 300 units respectively which is also suggested architecture in DDPG paper for the low-dimensional networks.

```
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."""
        x = F.leaky_relu(self.fc1(state))
        x = F.leaky_relu(self.fc2(x))
        return F.tanh(self.fc3(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))

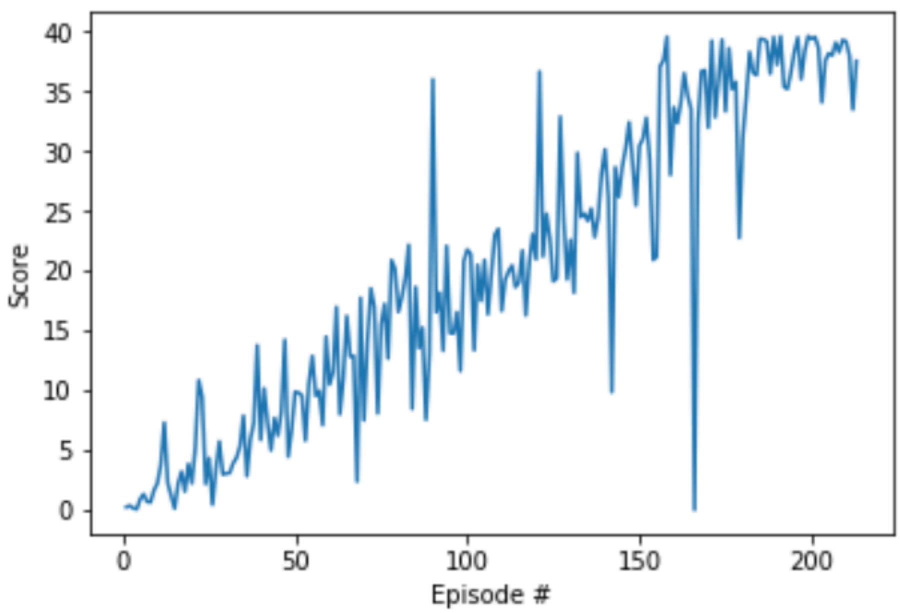
return self.fc3(x)

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) and finish the critic with logits.

REWARDS:

Please see the plot of the rewards .

```
Episode 100     Average Score: 9.44     Score: 21.76
Episode 200     Average Score: 28.73    Score: 39.37
Episode 213     Average Score: 31.12    Score: 37.53
Environment is solved!!!
```



IDEAS for FUTURE WORK :

- Probably there are better hyperparameters values, however I want to specifically test the different the neural architectures and activations. For example we are finishing the actor network with tanh but we are also np.clipping the action in the Agent class&act method after adding some noise. Maybe tanh is not required for the last layer of actor in this case.
- Also I didn't use batch-norm for the neural layers but it is suggested both in paper and in class notes.
- I think while using experience re-play; we can use prioritization or importance algorithm to sample more valuable experiences. That should also increase the performance of the agent.

Progress Videos;

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

Training:
https://youtu.be/iKWV0h8Qz-M

https://youtu.be/Q2CtsgBitZM

Smart Agent:

https://youtu.be/dG6t6Ef8LzE

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