Continuous Control

Abstract:

In reinforcement learning problems with overestimation as a result of function approximation in discrete action spaces such as going left, right..etc is well studied. The same issue appears with the actor-critic methods on continuous control domains. The paper: "Addressing Function Approximation Error in Actor-Critic Methods" addresses those issues.

"*In value-based reinforcement learning methods such as deep Q-learning, function approximation errors are known to lead to overestimated value estimates and suboptimal policies. We show that this problem persists in an actor-critic setting and propose novel mechanisms to minimize its effects on both the actor and the critic. Our algorithm builds on Double Q-learning, by taking the minimum value between a pair of critics to limit overestimation. We draw the connection between target networks and overestimation bias, and suggest delaying policy updates to reduce per-update error and further improve performance. We evaluate our method on the suite of OpenAI gym tasks, outperforming the state of the art in every environment tested*" (Fujimoto et al., 2018).

Data approach:

The typical data science project would require data sources. But since we are using a Reinforcement Deep Learning, the data is provided/acquired from the environment we are using. The environments used are obtained from the OpenAI Gym: pybullet-gym.

Models:

The project consists of 2 types of models: Actor and Critic. The Actor decides which action to be taken. The Critic decides the 'quality' of the action. The model we built is called Twin Delayed DDPG (Deep Deterministic Policy Gradient). We will be calling that the DDPG from now on. The DDPG consists of two layers. The first has a single Actor model and two Critics models. The second layer also has a single actor model and two Critics models. The first layer is called the Target Models while the second layer is called the Models.

Models architecture:

All these models have the same architecture of: 2 hidden layers of type fully-connected/dense. The first has 400 nodes and the second 300. They only differ in the input and output.

Actor: state -> 400 -> 300 -> output-> softmax -> actions

Critic: [state+action] -> 400 -> 300 -> Q_value

Optimizers:
All models are using ADAM optimizer.

Memory:
The DDPG is an off-policy model which means it learns from the past. To have a past a memory must be created. For that memory of 1000000 rows of(state, next_state, action, reward) was stored and its gets updated with new rows and deleting the old ones during training. Then from that memory, a selected size is passed back to the actor/critic models as a batch which is used to train.

Exploration vs Exploitation:
The model at first uses the random action sampling from the memory as it goes. Once 10000 iterations of the environment is done the model starts to decide action and learning.

The 10000 at the beginning is used to explore the space and to avoid getting stuck in a local minimum.

During training the model decides on actions using softmax to give the model a chance to explore actions and not get stuck on a single good action as that action might be producing a well enough result which is not the best but the model by chance, learnt it first.

To increase the exploration and bias in the model we add noise to action and finally to make sure the action of the model is within the environment's allowed limits we clamp the action.

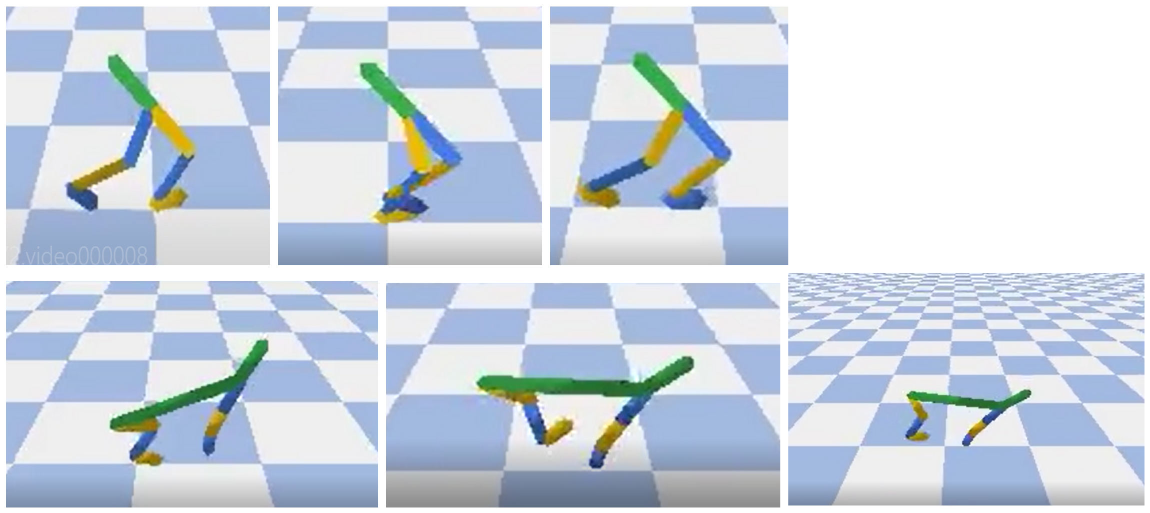
Execution:
The training starts with the Actor-Target. The AT gets the state of the environment and decides the action to be taken. Then the Two Critic Targets take in the future state(next_state) + The action made by the AT one step back(which is called next_action) Then each of the critics produces a Q value of which the lowest we will pick. We pick the lowest to not be optimistic and over estimate.

Then we use the current action of the state we are in as input into the two Critic models + the current state which would also produce two Q values. Then we use the min_Q value from one step back to calculate the loss = mse(Q1-min_Q)+mse(Q2-min_Q)

Finally we use the Q1 from the previous step to update the weights of the last model which is the Actor model.

As for updating the Target model: Actor and Critic. We will use the polyak update. Which slowly changes the weights of the Targets to the Models weights:
'tau' is the conversion percentage: (tau*model.param +(1-tau)*target_param)

After Training:



Tools:
Google Colab, Google Driver
Pytorch: torch, vision-torch
Numpy: numpy
OpenAi: gym
Pybullet: pybullet-gym