**Continuous Control Project**

To solve this project I used the distributed distributional deep deterministic policy gradient (D4PG) algorithm.

This algorithm is based on the deep deterministic policy gradient (DDPG) algorithm. Before implementing D4PG, I first made two extensions to the vanilla DDPG algorithm. First, I added gaussian noise to the actions played by the agent to encourage exploration. Secondly, I added batch-normalization layers in the actor and critic networks as described in: <https://doi.org/10.48550/arXiv.1509.02971>.

I found that the following hyper-parameter values worked well:

|  |  |
| --- | --- |
| **Hyper-parameter** | **Value** |
| Replay buffer size | 1e6 |
| Training batch size | 1024 |
| Discounting factor (gamma) | 0.99 |
| Soft update factor (tau) | 1e-3 |
| Actor learning rate | 1e-4 |
| Critic learning rate | 1e-3 |
| Weight decay | 0 |
| Agent updated every N steps | 50 |
| How many updates per update step | 40 |
| # units in first layer | 256 |
| # units in second layer | 128 |

Both actor and critic networks used the same number of units for their first and second layers. All but the final output layers were followed by a ReLU activation. The architecture of the actor and (distributional) critic networks is shown below:

**Critic**

**Actor**

action

state

state

batchnorm

batchnorm

a

batchnorm

Second layer

First layer

First layer

batchnorm

Second layer

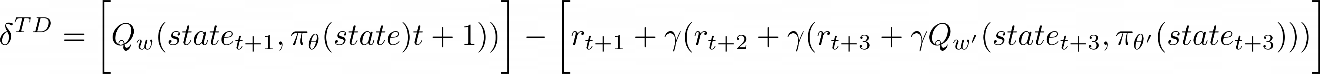
Mean, variance

action

From here, I implemented the D4PG algorithm by making the 4 major changes described in their paper (<https://arxiv.org/abs/1804.08617v1>).

First I used the Reacher environment with 20 agents acting in parallel, hence the distributed in D4PG. To implement this, a single actor network took in a batch of 20 observations at each timestep and output a batch of 20 actions (each action being 4 dimensional).

Secondly, I implemented a prioritized replay buffer such that experiences that were more ‘surprising’ to the agent had a higher probability of being replayed. These replay probabilities were computed to be proportional to the magnitude of the TD-error.

Thirdly, instead of using a 1 step TD return I implemented a 3 step TD-return as follows:

Where the ‘ differentiates target from local actor and critic networks.

An additional direction for the future would be to generalize this so that any N-step TD return could be used. Then we could search for the N-step return that results in the fastest learning.

Finally, I implemented a distributional critic. In this case, the critic does not output the state-action value function but instead outputs a distribution that approximates the state-action value function. In my implementation I approximate the state action value function using a single Gaussian distribution and for each state-action pair the critic outputs the mean and variance of the gaussian distribution approximating the return corresponding to this state-action pair. This critic was trained by minimizing the KL-divergence between the target and expected distribution using the 3-step Bellman operator (above equation). I found that the variance had to be clamped to always be greater than 0.25 or else KL divergence would diverge.

With this learning algorithm I see the agent learns within 40 training episodes and this learning is very stable (score is around +40) for the duration of 200 training episodes.

Training Rewards

Rectangle

Description automatically generated with medium confidenceThe agent learns to attain a score >+30 after about 40 episodes. This score is maintained for over 100 episodes and hence we considered the environment solved at 40 episodes.

Future Ideas

As mentioned above, implementing a generic N-step return or even using generalized advantage estimation may further improve how quickly the agent learns and is amenable to future investigation. Another area of future investigation would be using a mixture of Gaussian distributions to approximate the state-action value function instead of just using a single Gaussian.

In the future I also plan to train this agent in the more complex crawler environment. Specifically, I plan to train the agent using both DDPG and D4PG in this environment and observe any differences between these two learning algorithms in a more complicated environment.