Deep Reinforcement Learning Nanodegree

Deep Deterministic Policy Gradient (DDPG) in Action: Continuous Control

Ayham Zaza

Contents

Introduction

- An Agent **G** located at **State S**, interacts with its surrounding *environment* by trying a **Action A** to transfer to a **New State S'**. A **Reward R** is associated with each action; and the goal of the agent is to navigate the environment until it reaches an *Objective* **O** while achieving a maximum possible reward. At a given state, the agent *experience* is summarized by the tuple (**S,A,R,S'**).
- Actor-Critic Method are at the intersection of Value Based Methods (**DQN**) and Policy Based Methods (**REINFORCE**). It learns to estimate the Optimal Action Value Function $Q_{\pi*}(s, a)$ as well as parametrizing a Policy -usually stochastic- and learns to optimize it directly.
- Estimating Expected Return can be done in two ways:
    - *Monte-Carlo (MC) Estimate*: calculating the discount reward from a reward sequence of an episode. [High variance but unbiased]
    - *Temporal Difference (TD) Estimate*: uses a single reward sample and an estimate of the discounted total return the agent will get from the next state. [Low variance but biased]
- To reduce the variance, a base line is used.
- Actor-Critic Approach adjusts the probabilities of good and bad actions while utilizing a critic to tell good from bad actions more quickly. i.e. using a function approximation to learn a **Policy** and a **Value Function**
    - **Actor:** takes in a state, and outputs a distribution over actions
    - **Critic:** takes in a state, and outputs a state value function of policy π
- The Algorithm works as follows:
    - Input current state into the **Actor**, and get the action to take in that state.
    - Observe next state and reward to formulate experience tuple (**S,A,R,S'**).
    - Using TD-Estimate (Reward + Critic Estimate of next state, to train the **Critic.**
    - Calculate the **Advantage:** $A(s, a) = r + \gamma V(s', \theta_v) - V(s, \theta_v)$ from the Critic.
    - Train the actor, using the calculated **Advantage** as a base line.

- Deep Deterministic Policy Gradient **(DDPG**) is one realization Actor-Critic Methods. The **Critic**, however, approximates a maximizer over the **Q-Values** of the next state -not as a learned base line.

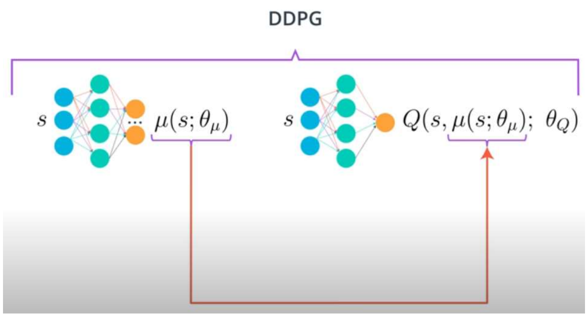

Figure 1: DRLND Lecture Notes. Udacity

Model Description

Network Architecture

- Actor-Critic DDPG network was implemented in PyTorch Linear module.

Actor Network:

```
#minimize variations in state sizes
self.norm_1 = nn.BatchNorm1d(state_size)
self.fc1 = nn.Linear(state_size, fc1_units)
self.fc2 = nn.Linear(fc1_units, fc2_units)
#self.dropout = nn.Dropout(0.5) # not useful,
self.fc3 = nn.Linear(fc2_units, 128)
self.fc4 = nn.Linear(128, action_size)
```

- A batch normalization layer was added to minimize variations
- Input: 24 nodes – corresponding to state size
- Output: 2 nodes – corresponding to action size (followed by tanh activation)
- To enable learning complex non-linear functions, the 4 hidden layers (fc1, fc2,fc3,fc4) are followed by ***relu*** activation function.
- (fc1,fc2) were both chosen to be of size 256. The remaining dimensions are in the figure above.

Critic Network:

```
self.norm_1 = nn.BatchNorm1d(state_size)
self.fcs1 = nn.Linear(state_size, fcs1_units)
self.fc2 = nn.Linear(fcs1_units+action_size, fc2_units)
self.fc3 = nn.Linear(fc2_units, 128)
self.fc4 = nn.Linear(128, 1)
```

- A batch normalization layer was added to minimize variations
- Input: 24 nodes – corresponding to state size
- Output: 1 nodes – corresponding to estimated Q_value
- To enable learning complex non-linear functions, the 4 hidden layers (fc1, fc2,fc3,fc4) are followed by ***relu*** activation function.
- (fc1,fc2) were both chosen to be of size 256. The remaining dimensions are in the figure above.

Hyperparameters

```
BUFFER_SIZE = int(1e5)  # replay buffer size
BATCH_SIZE = 128        # minibatch size
GAMMA = 0.99            # discount factor
TAU = 1e-3              # 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
```

The noise function plays an important rule in the speed of learning. Random Normal Noise performed well as compared to uniform noise. Thanks to the suggestions in the forum

Results

The decision for the number of layers/hidden units, as well the choice of Actor-Critic network solved the environment by achieving the target score in 96 episodes. figure_1

```
Episode 96      Average Score: 30.062
Environment solved in 96 episodes!     Average Score: 30.062
```

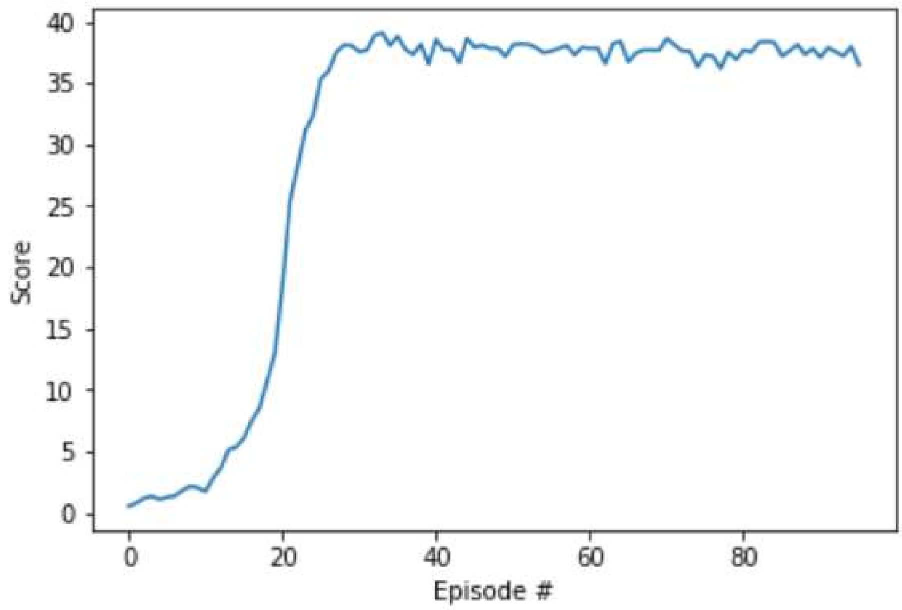

Figure 2: A plot of rewards per episode: the agent is able to receive an average reward (over 100 episodes) of at +13

Future Improvements

- Trying different network architectures
- Aiming Solving the environment with less time
- Solving the environment with higher average reward
- Trying different learning learning algorithms.
    - Asynchronous Actor-Critic Agents (A3C)
    - Trust Region Policy Optimization (TRPO) and
    - Proximal Policy Optimization (PPO)

References

- Udacity Deep Reinforcement learning Nanodegree Program. Lecture Notes and Videos and student Forum (https://www.udacity.com/course/deep-reinforcement-learning-nanodegree--nd893)
- https://github.com/udacity/deep-reinforcement-learning
- Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. R Lowe, et al. 2017
- Trust Region Policy Optimization. John Schulman et al. 2017
- Reinforcement Learning: An Introduction. Book by Richard S. Sutton and Andrew Barto. Second Edition. 2018