Report

May 28, 2020

1 Continuous Control

1.0.1 Introduction

Unity ML-agent library has a lot of simulation evnrionments to test intelligent agent models. This notebook solves one of the envrionnment with a task name called **Reacher**. The task is to move a 2-joint robot arm to a specific position. As the robot arm correctly follows the guided position, more points can be obtained. In this simulation, the agent uses a Deep Deterministic Policy Gradient model (DDPG) to learn an intelligent behavior to meet the objective and the environment will have 20 agents of robot arms. All the code is self contained for a potential modification of any logic. Each component is separated to its own cell, so it can be easily written to a separate file using "%writefile" if necessary

---

1.0.2 Environment

Requirements

- Linux based OS
- Python (dependency is listed in requirements.txt)
- Linux 20-arm Reacher environment

The environment spec

- Number of agent: 20
- State space: 33 dimensional vector (for each agent)
- Action space: 4 numbers of range [-1, 1]
- Reward: +0.1 (If the agent's arm is located at a guided location)

1.0.3 Implementation

ActorModel

- **Description**
  - Standard fully-connected NN with a Batchnorm layer and ReLU activation. This model is used to model the deterministic policy, so it outputs 4 continous action vectors.
- **Architecture**

  - 2 Hidden layers (128, 64 nodes)
  - 1 Output layer (4 nodes)
  - Each hidden layer uses ReLU and Batchnorm1d
  - Final output layer is a single linear layer with tanh activation to conform to the action value range
- **Discussion**
  - The initial hidden layer expanded the number of nodes for exploring rich feature space
  - ReLU and Batchnorm layer are used for stable training
  - Final output layer is combined with tanh to generate action of a correct range

CriticModel

- **Description**
  - A fully-connected NN which combines a state and action to generate Q value for a given state, action pair.
- **Architecture**
  - 2 Hidden layers (128, 128 nodes)
  - 1 Output layer (1 node)
  - Each hidden layer uses ReLU
  - State value is normalized by BatchNorm1d
  - Final output layer is a single linear layer without any activation function
- **Discussion**
  - To deal with different characteristics of state and action, they're embedded to the first hidden layer separately
    - The first hidden layer 128 nodes are divided according to the ratio of state and action dimensions (33 to 4)
  - The initial hidden layer expanded the number of nodes for exploring rich feature space
  - The number of nodes are decided with experiments
  - Batchnorm layer is removed because it gives worse performance
  - Final output layer directly generates an action value

Buffer

- **Description**
  - Replay buffer used in a DDPG algorithm. It is a standard deque with a sampling function
- **Interface**
  - Initialization
    - sz: Replay buffer size
    - pf: Probability function for sampling
  - put(obj): Stored obj in a replay buffer
  - sample(n): Sample n objects from a replay buffer
  - update_pf(pf): Update the probability function pf
- **Discussion**
  - pf can be used for prioritized experience replay
  - It was used to explore the potential benefit of the skew in experience sampling, but was default to uniform sampling

Training

- **Description**
  - Standard DDPG training algorithm
- **Detail**
  1. Initialize the models(2 actors, 2 critics) and hyperparameters
  2. The current-actor-model (agent) interacts with the **Reacher** Unity environment to collect tuples (state, action, reward, next-state)
     1. Exploration in continuous space is implemented using a Gaussian noise. The noise's standard deviation is controlled by an epsilon parameter
     2. Due to the probabilistic nature of the noise, the action is clipped to range [-1, 1]
  3. The collected tuples (experiences) are stored in a replay buffer
  4. The experience is randomly batch-sampled every 20 steps("train_step") and used to train the NN model
     1. Due to the multi-agent nature of the environment, The single batch size is 512 and 10 successive gradient updates happen in a single training procedure.
     2. The training procedure follows a standard DDPG algorithm
        1. Current-critic-model is trained with TD-update using Target-actor/critic-models
        2. Current-actor model is trained with Q value using Target-critic-model
        3. Target-actor/critic-models are soft-updated with current models using a small mixture ratio(tau)
  5. 2-4 are repeated until the average score(return) reaches the average objective score (30)
- **Hyperparameters**

name	value	description
n_episodes	500	Max number of episodes to explore
max_t	3000	Max number of time steps of each episode
bn	512	Training single batch size
gamma	0.99	Return discounting factor
eps, eps_decay	0.25, 0.99	Parameter to control noise standard deviation (for exploration)
tau, tau_decay, tau_min	0.001, 0.9999, 0.001	Target network soft-update parameters
train_step	20	Frequency of the training
train_cnt	10	Number of gradient-updates in a single training procedure

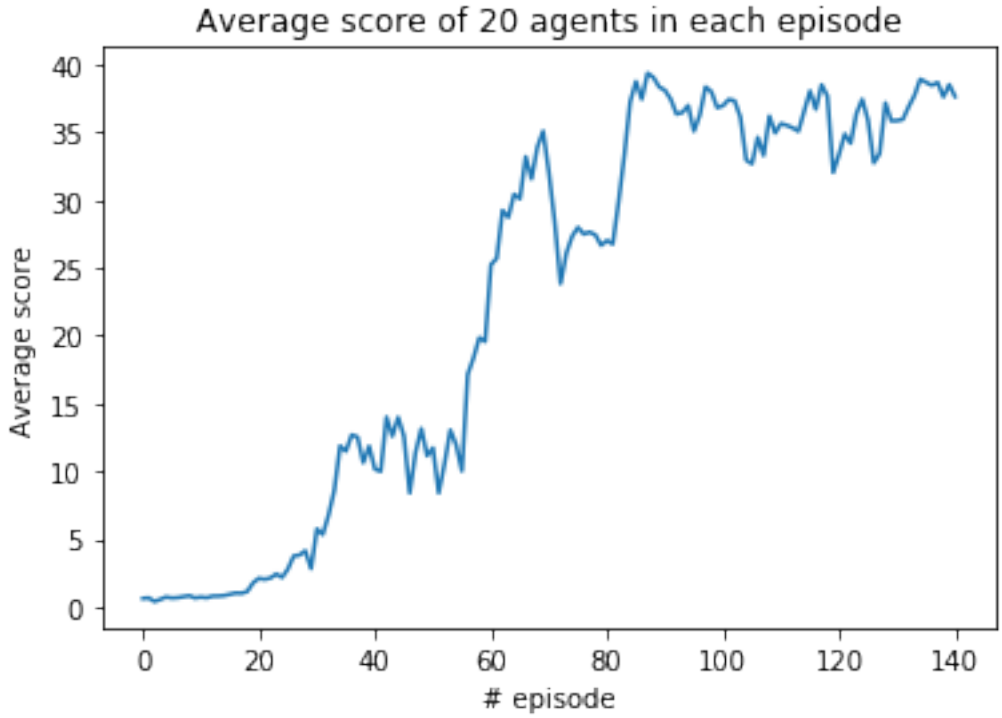
1.0.4 Results

The following chart shows the average score of 20 agents in each episode as it trains. It achives ~30 for 100-episode average and ~40 for single episode, within ~140 episodes

A few interesting observations are

- There is a sequence of episodes when the average score is quite static. Its interpretaion in a real world could be the required time to learn new skills.
- After the static time, there is a significant increase in the average score, this can be another proof that a meaningful skill was learned and utilized to solve the environment

[23]: `Text(0,0.5,'Average score')`



1.0.5 Future work

1. Batchnorm layer is usually known to help the NN optimization, but it seemed to worsen the performnace with the given NN-architecture and task/environment. A few hypotheses can be explored with a detailed look at the training steps
    - The reward is quite big in this environment. With a simple NN-architecture heavily using BN, one signle output layer needs to deal with the big reward scale, which might not be appropriate
    - BN might not help much in RL setting where the input data has a very big variance. (considering the usual batch size of 32 ~ 256 time-steps)
    - NN is too small and simple. BN might not have a big impact in a small NN
2. In DDPG algorithm, the actor can be trained with a Q-value estimate of critic current or target network. It can be expected that the current encourages faster training and the target can help more stable training. This expectation can be tested with an experiment and can be used to train a model faster and more reliably.

3. Other policy-gradient methods can be tried and compared against DDPG algorithm