Demonstrate that a Distributed Supply Chain Problem can be Managed by Co-Operating AI Agents

Component 3

6th November 2020

1 Task

Using Machine learning to build a model that makes the same agent decisions . The repository for all the code is available on Github

2 Team

THE_OUTLIERS:
Apoorva Vikram Singh
Divyansha
Anubhav Sachan
Asis Kumar Roy
Yash Srivastava

College: National Institute of Technology Silchar

3 Recapitulation

In the previous task, we defined reward function, environment, states, actions, transition function and demand function. We also defined a baseline policy and optimized it using brute force methods. Now, we will proceed to use DDPG algorithm in order to train our agent to make the decisions.

4 Deep Deterministic Policy Gradient Algorithm

Deep Deterministic Policy Gradient (DDPG) algorithm focuses on simultaneously learning a Q-function and a policy and is most suitable for continuous control problems which fits perfectly with our problem statement. It utilizes the Bellman equation and off-data policy to determine the Q-function. It further uses the Q-function to determine the policy.

DDPG algorithm is closely related to Q-learning and has the same motivation i.e. if the optimal action-value function $Q^*(s, a)$ is known then the optimal action $a^*(s)$ in any given state can be determined by solving the following equation:

$$a^*(s) = \underset{a}{\operatorname{argmax}}\, Q^*(s, a)$$

Due to the presence of a continuous action space exhaustive evaluation of the space is not possible. Therefore an efficient, gradient-based learning rule utilized for a policy $\mu(s)$ that exploits the continuous action and presumably differential Q-function. Thus instead of running an expensive optimization to compute $max_a Q(s, a)$, we are able to approximate it to be nearly equal to $Q(s, \mu(s))$.

4.1 Q-learning in DDPG

The bellman equation serves as a starting point for learning an approximation to $Q^*(s,a)$. The equation has been described below:

$$Q^*(s,a) = \underset{s' \sim P}{\mathrm{E}} \left[r(s,a) + \gamma \max_{a'} Q^*(s',a') \right]$$

Here, $s' \sim P$ is just a shorthand for saying that the next state, s', is sampled by the environment from a distribution $P(.|s,a)$.

Consider the approximator to be a neural network $Q_\phi(s,a)$, with ϕ parameters. If there is a set D of transitions (s,a,r,s',d), here d tells whether the s' state is terminal or not. A mean-squared Bellman error (MSBE) function can be defined as:

$$L(\phi, D) = \underset{(s,a,r,s',d) \sim D}{\mathrm{E}} \left[\Big(Q_\phi(s,a) - \big(r + \gamma(1-D) \max_{a'} Q_\phi(s',a') \big) \Big)^2 \right]$$

This function tells us how closely Q_ϕ comes to satisfying the Bellman equation.

The Q-learning algorithms focus on the minimization of MSBE (Eq. 4.1) and hence, is usually employ either of the two popular methods:

- Target Networks: The algorithm introduces a term called defined as: *target*

  $$r + \gamma(1-d) \max_{a'} Q_\phi(s',a')$$

  The aim of using the target is to make the original q-function similar to the target and hence, minimizing the error. But, since, the parameters of Q-value function and the target function are same (i.e. ϕ), the MSBE minimization becomes unstable. Hence, the target network is composed of new set of parameters (ϕ_{target}), which come close to ϕ, but with a time delay (lags behind). The target parameters are simply copied over from the main equation, and updated once per main network update by polyak averaging.

- Using Replay Buffers: Replay Buffer $\mathcal{D}$ is a set of previous experiences composed of (s,a,r,s',d). To have a stable behaviour, the $\mathcal{D}$ should be large enough to contain a diversified (limited) range of experiences. This set should not have most recent data which will overfit the model and if it is quite large, it'll hamper the learning pace.

4.2 Policy Learning in DDPG

In order to learn a deterministic policy $\mu_\theta(s)$ that proves us with an action which maximizes $Q_\phi(s,a)$ gradient ascent (with respect to policy parameters) is performed to solve:

$$\max_\theta \underset{s \sim D}{\mathrm{E}} [Q_\phi(s, \mu_\theta(s))]$$

4.3 Algorithm

Algorithm 1: Deep Deterministic Policy Gradient Algorithm

Input: initial policy parameters θ, Q-function parameters ϕ, empty replay buffer D

Set target parameters equal to main parameter $\theta_{targ} \leftarrow \theta$,$\phi_{targ} \leftarrow \phi$;

while *no convergence* **do**

  Observe state s and select action a = $clip(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim N$;

  Execute a in the environment;

  Observe next state s', reward r and done signal d to indicate whether s' is terminal

  Store (s, a, r, s', d) in replay buffer D

  If s' is terminal, reset environment variable

  if *it is time to update* **then**

    for *however many updates* **do**

      Randomly sample a batch of transitions, B = (s, a, r, s', d) from D

      Compute targets

$$y(r, s', d) = r + \gamma(1 - d)Q_{\phi_{targ}}(s', \mu_{\theta_{targ}}(s'))$$

      Update Q-function by one step of gradient descent using

$$\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d)\epsilon B} (Q_\phi(s, a) - y(r, s', d))^2$$

      Update policy by one step of gradient descent using

$$\nabla_\phi \frac{1}{|B|} \sum_{s\epsilon B} Q_\phi(s, \mu_\theta(s))$$

      Update target networks with

$$\phi_{targ} \leftarrow \rho\phi_{targ} + (1 - \rho)\phi$$

$$\theta targ \leftarrow \rho\theta targ + (1 - \rho)\theta$$

    end

    ;

  end

end

5 Results

We trained the DDPG trainer from Ray-Project[1] for 300 epochs, and observed that maximum mean reward (i. e. profit) to be Rs. 9418.82 which is significantly

[1]Ray-Project utilizes the distributed computing systems and helps to speed up the training process.

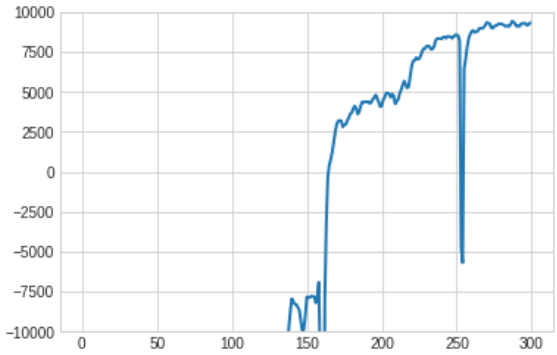

Figure 1: Episode Mean Reward

improved value over the baseline (s, Q) policy's Rs. 4758.12, as shown in fig. 1. The descriptive progress .csv file contains all the information iteration wise. We also observed that, the Q-value function was converging and the maximum mean Q-value as found to be 4171.131. All the experimentation for DDPG Algorithm were performed on Google Colab for its GPU support.

6 Conclusion and future work

The environment setup has been done in order to simulate a supply chain optimization problem. We have simulated a baseline policy and optimized it using brute force techniques. We have then optimized it using the DDPG algorithm which significantly outperformed our baseline.