IMPORTANCE OF DATA IN OFFLINE REINFORCEMENT LEARNING

Marco Enzo Squillacioti, Steinn Orri Erlendsson, & Shubham Sanjay Ingole

s200142, s153716, s200092

ABSTRACT

Offline learning algorithms have great potential to allow massive data sets to be turned into effective decision makers. Efficient offline reinforcement learning approaches will be able to derive policies that are as useful as possible from the available data [1]. As in the real-world implementations have significant consideration for off-policy reinforcement learning using a defined offline dataset for recorded interactions [2].

This paper will showcase the challenges and importance of data in implementing offline reinforcement learning using cart-pole balance problem. To identify these challenges, a comparison is made between online and offline training by using same parameters. To introduce a baseline and to generate dataset, we adapt algorithm from OpenAI Spinning Up - Deep Deterministic policy gradient (DDPG) [3] and MinimalRL [4] implementations in PyTorch. The dataset used by the offline algorithm originates by recording states, actions and rewards from the online implementation. The article also demonstrates different experiments with the recorded datasets in offline RL to achieve similar performance as online RL. The implementation can be found in the following GitHub repository https://github.com/marcosquilla/On-OfflineDDPG.

1. INTRODUCTION

Mathematical formalism for learning-based regulation is offered by reinforcement learning that performs effective decision making in real world [2]. This improved learning, will be able to build optimal behavioural capabilities, represented by policies, that can clarify user-specific operations, where the policy chooses an action and the reward function defines what the agent will obtain from performing such action [2]. This methodology of reinforcement learning can be referred directly with decision-making problem and with the objective to optimise sustainable performance of a complex and changing environment that responds to each decision.

This article's aim is to implement an algorithm and try to identify the challenges in offline RL aligned by varied dataset, by using a cart pole balance problem from the deep-mind control suite. Offline Reinforcement Learning relies on training a policy on a static dataset without the ability to explore the environment. Because the algorithm is essentially identical, the dataset is the main focus of attention. Thus, this paper focuses on matching the performance of Offline with Online RL by training with the same hyper-parameters in both cases and varying the dataset used.

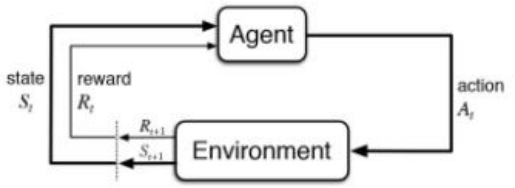

Fig. 1: Classic architecture of reinforcement learning [5]

The dataset is generated by training different off-policy RL algorithms, along with using sampling techniques on the obtained datasets. Interestingly, the results illustrate that the offline algorithm performs worse than the online algorithm. Furthermore, this paper aims to perform experiments with the generated datasets to identify challenges and achieve the desired results.

2. RELATED WORK

Looking at the framework in section [1], the reinforcement learning's application can be used for solving complex problems. The limitations of the Online application are due to the interaction of an agent with an online environment, where the agent can collect sufficient data for learning every task. Moreover it interacts with the environment using a partially trained policy which can take expensive or unsafe actions [2].

In contrast, offline RL faces the challenges in distributional shift and learning a policy from the fixed dataset, by not interacting with the environment [2]. This methodology can leverage a large amount of existing recorded interactions for solving decision-making problems such as robotics and healthcare [2].

Moreover, the below methodologies of reinforcement learning will be discussed to address the developed dataset and to examine the challenges. Fig 2 depicts the difference between online, off-policy, and offline reinforcement learning [1]. Whereas, in (a) Online RL, the policy is update on every interaction with the environment, in the second framework (b), the off-policy, the interactions are recorded in a buffer

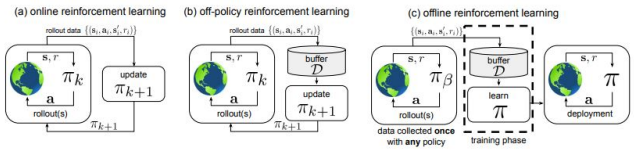

Fig. 2: (a) online RL, (b) off-policy RL with online data collection, and (c) offline RL [2]

(which can have a maximum size), and every a fixed number of episodes the policy is update with using this buffer. Lastly, in (c) the Offline RL trains solely on data that was previously collected by another entity (be it another algorithm or real-world interactions) without any additional online data collection [5]. Here, the implementation mainly focuses on Off-policy and Offline Reinforcement learning.

3. METHODOLOGY

This section is highly influenced by reference [3]
It gives an introduction to the gradient based learning methods used in DDPG to utilise Q-learning with continuous action spaces.

3.1. Deep Deterministic Policy Gradient (DDPG)

DDPG is an off-poicy algorithm. It uses off-policy data and the Bellman equation to learn a Q-function. From the Q-function the algorithm learns the policy. It is an algorithm that allows to perform deep Q-learning in continuous action spaces such as the cart pole balance problem.

3.2. Discrete vs. Continuous action spaces

In Q-learning we can calculate the optimal action-value $a^*(s)$ from the optimal action-value function

$$a^*(s) = \arg\max_a Q^*(s,a) \tag{1}$$

for any given state s. Now when the action space is continuous the computations become infeasible.
The essence of DDPG is that it assumes that the function $Q^*(s,a)$ is differentiable as the action space is a matter of fact continuous it must be differentiable with respect to the action. This leads to the fundamental part where this idea is utilised to create a gradient-based learning rule for a policy $\mu(s)$. So now the optimal action is approximated as the action picked by the policy.

$$\arg\max_a Q(s,a) \approx Q(s,\mu(s)) \tag{2}$$

3.3. The Bellman Equation for Q-learning in DDPG

The optimal value function $Q^*(s,a)$ is given by the following Bellman equation.

$$Q^*(s,a) = \underset{s'\sim P}{E}\Big[r(s,a) + \gamma \max_{a'} Q^*(s',a')\Big] \tag{3}$$

Where $s' \sim P$ stands for *s'* is the next state sampled from a distribution $P(\cdot|s,a)$, $\gamma \in \{0,1\}$ is a discount factor. So the optimal value function is estimated in the Bellman equation in 3 with the expected return of the reward of the current state plus the estimated maximum reward of the next state *s'* times the discount factor γ.

3.3.1. Mean Squared Bellman Error in Q-learning

In Q-learning the approximate optimal value function is a neural network, $Q_\phi(s,a)$ with ϕ parameters.
In this research a dataset of $\mathcal{D}$ of transitions (s,a,r,s',d) is generated. The *d* is a binary variable taking the value of 1 if *s'* is terminal and 0 otherwise. This variable makes it possible to calculate the *MSBE* as shown in equation 4.

$$L(\phi,\mathcal{D}) = \underset{(s,a,r,s',d)\sim\mathcal{D}}{E}\Big[\Big(Q_\phi(s,a) - (r + \gamma(1-d)\max_{a'} Q_\phi(s',a'))\Big)^2\Big] \tag{4}$$

This gives an approximation of how close $Q_\phi(s,a)$ is to satisfying the Bellman equation. When minimising the *MSBE* we are in essence getting the optimal value function as close as possible to the value of the target:

$$r + \gamma(1-d)\max_{a'} Q_\phi(s',a')) \tag{5}$$

This causes a problem as the target is dependent on ϕ which are the values being trained in the Q-function. In Q-learning it is generally dealt with by creating a second network ϕ_{targ} with similar parameters called the target network. The target networks parameters are updated every time the main network is updated with the following process called polyak averaging.

$$\phi_{targ} \leftarrow \rho\phi_{targ} + (1-\rho)\phi \tag{6}$$

$\rho \in [0,1]$ and usually closer to 1 is a hyperparameter.

3.3.2. Mean Squared Bellman Error in DDPG

As the action space is continuous a neural network is initialised to learn the policy $\mu_\theta(s')$ with the parameters θ.

$$L(\phi,\mathcal{D}) = \underset{(s,a,r,s',d)\sim\mathcal{D}}{E}\Big[\Big(Q_\phi(s,a) - (r + \gamma(1-d)Q_{\phi_{targ}}(s',\mu_\theta(s')))\Big)^2\Big] \tag{7}$$

But that has the same problem as parameters ϕ so another policy network with parameter θ_{targ} which is updated the same way as ϕ_{targ}.

$$L(\phi,\mathcal{D}) = \underset{(s,a,r,s',d)\sim\mathcal{D}}{E}\Big[\Big(Q_\phi(s,a) - (r + \gamma(1-d)Q_{\phi_{targ}}(s',\mu_{\theta_{targ}}(s')))\Big)^2\Big] \tag{8}$$

3.4. Online vs. Offline Reinforcement learning

When comparing online and offline RL environments, it is understood that on-policy data should be accessed from the environment as it confirms a process in which agents choose behavior and decide to contribute to high rewards and then collect input to correct their errors [2]. To investigate further, the following DDPG algorithm is used to set the online and offline RL in order to showcase the two distinct scenario with different dataset[6].
Note: In the implementation of both algorithms in Python τ was used ($\tau = 1 - \rho$).

Algorithm 1 Deep Deterministic Policy Gradient - *Online*

Input initial policy parameters θ , Q-function parameters ϕ, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\theta_{targ} \leftarrow \theta, \phi_{targ} \leftarrow \phi$
repeat
  Observe state s and select action $a_{clip}(\mu_\theta(s = +\epsilon, a_{Low}, a_{High}$, where $\epsilon \sim \mathcal{N}$
  Execute a in the environment
  Observe next state s' is terminal
  Store (s,a,r,s',d) in replay buffer $\mathcal{D}$
  If s' is terminal, reset environment state.
  for however many updates **do**
    Randomly sample a batch of transitions, $B = ((s, a, r, s', d)$ from $\mathcal{D}$
    Compute targets $y(r, s', d) = r + \gamma(1 - d)Q_{\phi_{targ}}(s'))$
    Update Q-function by one step of gradient descent using $\nabla_\phi \frac{1}{|B|} = \sum_{(s,a,r,s',d) \in B)}(Q_\phi(s, a) - y(r, s', d))^2$
    Update policy by one step of gradient ascent using $\nabla_\theta \frac{1}{|B|} = \sum_{s \in B} Q_\phi(s, \mu_\theta(s))$
    Update target networks with $\phi_{targ} \leftarrow \rho\phi_{targ} + (1 - \rho)\phi$ $\theta_{targ} \leftarrow \rho\theta_{targ} + (1 - \rho)\theta$
  end for
until convergence

Algorithm 2 Deep Deterministic Policy Gradient - *Offline*

Input initial policy parameters θ , Q-function parameters ϕ, load data $\mathcal{D}$
Set target parameters equal to main parameters $\theta_{targ} \leftarrow \theta, \phi_{targ} \leftarrow \phi$
  for however many updates **do**
    Randomly sample a batch of transitions, $B = (s, a, r, s', d)$ from $\mathcal{D}$
    Compute targets $y(r, s', d) = r + \gamma(1 - d)Q_{\phi_{targ}}(s'))$
    Update Q-function by one step of gradient descent using $\nabla_\phi \frac{1}{|B|} = \sum_{(s,a,r,s',d) \in B)}(Q_\phi(s, a) - y(r, s', d))^2$
    Update policy by one step of gradient ascent using $\nabla_\theta \frac{1}{|B|} = \sum_{s \in B} Q_\phi(s, \mu_\theta(s))$
    Update target networks with $\phi_{targ} \leftarrow \rho\phi_{targ} + (1 - \rho)\phi$ $\theta_{targ} \leftarrow \rho\theta_{targ} + (1 - \rho)\theta$
  end for

4. EXPERIMENTS AND RESULTS

4.1. The Data

Looking into the experiments, The data was generated with the Online version of DDPG algorithm. The algorithm generates its own data and learns by the data generated from it. Where, the data is then stored as a sample for the Offline algorithm to execute futher.

The data was generated in two ways. One where the algorithm was executed *without random actions and the result was then stored*. Indexes referred to as *D1*. The other method was running the similar algorithm *with random actions and storing those results*, Indexes referred as *RA*.

4.1.1. Different Samples of the Datasets

Generating the two different datasets proceeds to perform three different sampling methods that were used for *D1*.

1. The whole dataset was used
2. Only first 4000 episodes were used
3. Balanced dataset based on a uniform sample of rewards *r* on 80 equally sized intervals in the continuous range min *r* : max *r*.

Where, Dataset *RA* was not modified in any way.

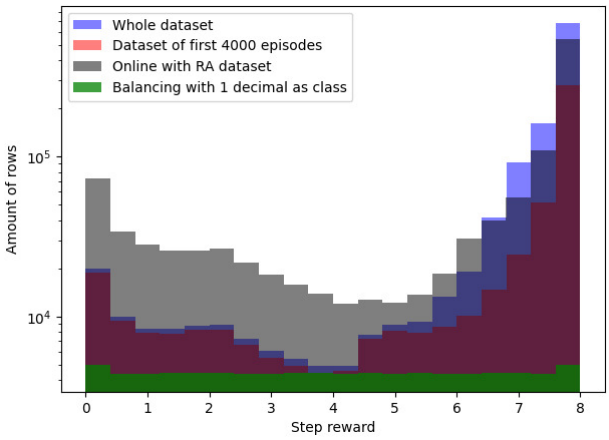

Fig. 3: Histogram of recorded actions based on rewards

Considering the second method, the value of 4000 was not randomly chosen. When looking at the reward graph from the online algorithm (Fig. 5), it can be seen that it is at this point when the algorithm stops improving it's performance *(Note that, in the online algorithm, for every episode there are 20 updates to the functions)*.

In Figure 3 we can see how the blue (*D1*) and red (first 4000 episodes of *D1*) are skewed to the higher values of the

reward. Whereas, the grey (*RA*) has more of a uniform distribution. On the other hand, the green (Balanced dataset) has a uniform distribution of rewards but has a low number of samples compared with the others.

4.1.2. Diversity of Observations

To better visualise the diversity between datasets a PCA decomposition of *states+actions* are illustrated against the reward in Figure 4. It is interesting to see how all datasets have a higher variance of the PCA when the rewards are closer to zero. Particularly interesting to see the shift when the reward is higher than 4. As if any reward above 4 has less diverse observations than those of lower rewards.

It is apparent that the *RA* dataset has higher variance of the PCA decomposition than the *D1* dataset and *D1* has higher variance than the two datasets created from it, the *balanced dataset* and *dataset of first 4000 episodes*.

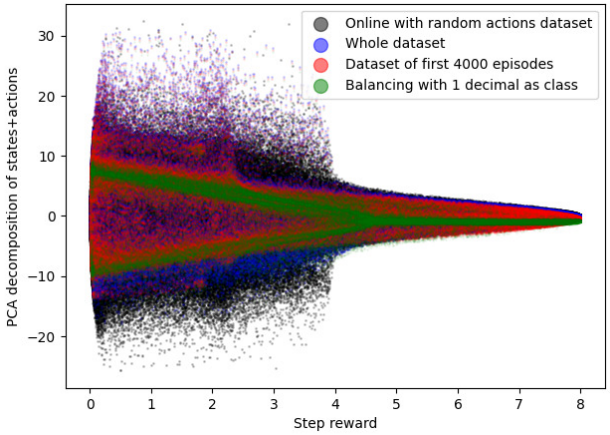

Fig. 4: Diversity of observations and actions versus reward obtained on different datasets

4.2. Performance of *Offline-* vs. *Online-DDPG*

Evaluating the performance of the *Offline DDPG* the total evaluation reward of the algorithm is compared with that of the *Online DDPG* algorithm that was used to generate the data.

When looking at the unaltered online algorithm in figure 5 and how it's dataset performs in the offline, it can be seen that the performance is very poor when using the whole dataset. This was attributed to a general lack of variety of actions, as after the first 4000 episodes the online algorithm already masters the problem, leaving the following 6000 episodes only with samples of very good actions and rewards.

The offline algorithm needs to learn both good and bad actions. Given that it samples randomly from the whole dataset, it is much more likely that it'll pick good actions than bad actions. Therefore, two approaches were taken to tackle this issue: take all the episodes until the online algorithm reaches the plateau and randomly balance the dataset.

It is evident that taking the data from the first 4000 episodes hurt the performance, while balancing the dataset improved it marginally. The reason for this is that, in both methods, even though the dataset used in more balanced rewards-wise, records are being removed, which hurts the diversity of states and actions. This is a key factor in the offline algorithm, as this diversity is how it *explores* the environment.

This is the reason for the *RA* algorithms. By introducing random actions in the online algorithm, a great variety is included (as it can be seen in figure 4). The probability of taking these random actions, as opposed to the ones calculated by the policy, increases linearly from 0% at the start of the training until 50% at the end (the reason for making it increase rather than decrease is that by hurting the training at the beginning the policy will never achieve any training at all). As it can be seen in figure 5, the performance is sub par when compared to the regular online algorithm and it continues to decrease as the training progresses.

This new dataset includes a variety of actions unparalleled by any of the other methods, while being also relatively well balanced rewards-wise. Figure 5 shows what this translates into in the offline algorithm: very similar training speed, slightly higher reward and better consistency when this reward is obtained. It is worth noting that the offline had drops in performance at seemingly random points during training (in figure 5 particularly: in the area around 150000 updates) in every experiment run. This is easily solved by saving the parameters of the network when the reward obtained is the maximum, instead of always saving the latest episode.

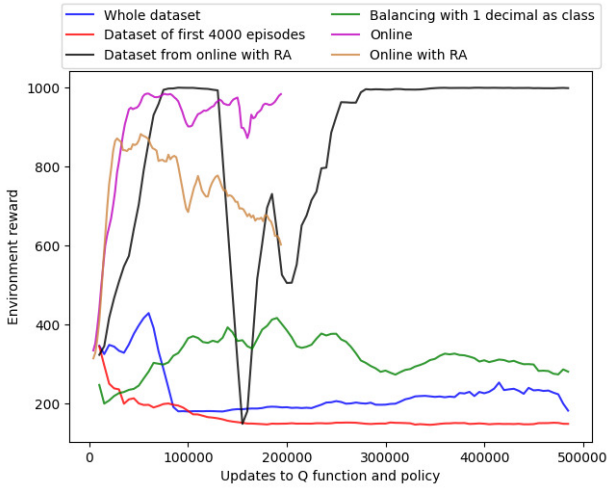

Fig. 5: Total evaluation reward of the different algorithms during training.

Hyperparameters of Online Algorithm $\gamma = 0.999$, $\tau = 0.01$, learning rate $\mu_\theta = 0.0001$, learning rate $Q_\phi = 0.001$.

Hyperparameters of Offline Algorithm $\gamma = 0.999$, $\tau = 0.01$, learning rate $\mu_\theta = 0.00001$, learning rate $Q_\phi = 0.0001$.

5. CONCLUSION

This paper examines the Offline reinforcement learning on cart pole balance problem based on the DDPG algorithm. This article demonstrates how the diversity of the data and randomness added to the action space affects the performance of both the *Online* as well as the *Offline* DDPG algorithms. It also indicates the importance of having diverse data as the offline algorithm can not interact with an environment and is entirely dependent on the observations it has to train with. These experiments help to identify the challenges and limitation in reinforcement learning.

It is, however, hard to imagine how the included randomness can be obtained when data is obtained from the real world. The better alternative in this case seems to be to gather a much higher amount of data and, if possible, from different sources. Different sources could provide different ways of obtaining good rewards, allowing the offline algorithm to train from this diversity.
Because the cart pole balance is such a simple problem, in which a very limited amount of actions can return good results, it is tricky to simulate different sources and it is likely that most well trained policies behave in a very similar matter. In a much complex environment this will not occur, thus a good diversity of data sources could mitigate the lack of diversity.
If it is not possible to have several sources or they do not contribute to the diversity of the dataset, introducing some balancing can be worth a try. In this paper, balancing based on reward obtained was considered, but the evidence found shows that the algorithm would benefit from including the actions and states in the process. In this way, not only the resulting dataset will be balanced, but diversity will be preserved. For this task, a Principal component analysis (PCA) could be used or, more interestingly, use the latent space from an variational autoencoder (VAE).

6. REFERENCES

[1] George Tucker Justin Fu Sergey Levine., Aviral Kumar, "Offline reinforcement learning: Tutorial, review, and perspectives on open problems," 2020.

[2] Mohammad Norouzi Rishabh Agarwal, Dale Schuurmans, "An optimistic perspective on offline reinforcement learning," 2020.

[3] Josh Achiam, "Open ai spinning up – deep deterministic policy gradient," 2018.

[4] Seungeunrho, "Minimalrl, github repository," .

[5] Sergey Levine, "Decisions from data: How offline reinforcement learning will change how we use machine learning," .

[6] jachiam, "Open ai, spinning up, github repository," .