Deep Deterministic Policy Gradients, Variants of DQN

Easswar Subramanian

TCS Innovation Labs, Hyderabad

Email : easwar.subramanian@tcs.com / cs5500.2020@iith.ac.in

October 25, 2022

Overview

1. Review
2. Deterministic Policy Gradient Algorithm
3. Double DQN
4. Priortized Experience Replay
5. Practical Tips

Review

Different Policy Gradient Formulations

Gradient of the performance measure is given by

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(a_t|s_t) \Psi_t \right]$$

1. $\Psi_t = \sum_{k=0}^{\infty} \gamma^k r_{k+1} = G_0$, Total reward of the trajectory
2. $\Psi_t = \sum_{t'=t}^{\infty} \gamma^{t'} r_{t'+1} = G_{t:\infty}$,Total reward following action a_t
3. $\Psi_t = \sum_{t'=t}^{\infty} \gamma^{t'} r_{t'+1} - b(s_{t'}) = G_{t:\infty} - b(s_t)$, Baseline version of the previous formula
4. $\Psi_t = \gamma^t Q^{\pi_\theta}(s_t, a_t)$, State action value function
5. $\Psi_t = \gamma^t A^{\pi_\theta}(s_t, a_t) = \gamma^t \left[Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t) \right]$, Advantage function
6. $\Psi_t = \gamma^t \left[r_{t+1} + \gamma V^{\pi_\theta}(s_{t+1}) - V^{\pi_\theta}(s_t) \right]$, TD residual

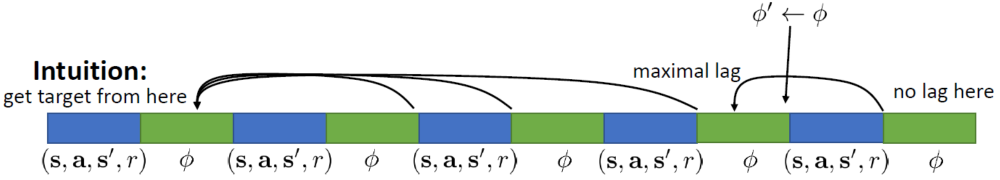
Deterministic Policy Gradient Algorithm

DQN Algorithm

Algorithm DQN Algorithm

Intialize replay memory D to capacity N
Initialize action value function Q with parameters ϕ
Initialize target action value function $\widehat{Q}$ with parameters $\phi' = \phi$
for episodes $= 1$ to M **do**
    Initialize start state s_1
    for steps $t = 1$ to T **do**
        Select action a_t using ϵ-greedy policy
        Execute action a_t and store transition (s_t, a_t, r_t, s_{t+1}) in D
        Sample random minibatch (size B) of transitions from D
        for b $= 1$ to B **do**
            Calculate targets for each transitions (Bellman backup or reward)
        end for
        Perform a gradient descent step on $(y_i - Q_\phi(s_t, a_t))^2$ w.r.t ϕ
        Every C steps set $\widehat{Q} = Q$
    end for
end for

Alternative Target Network



Polyak Averaging

- Replace target network update step (Step 14) by

$$\phi' : \phi' \leftarrow \tau\phi' + (1-\tau)\phi$$

- Typical value for $\tau = 0.99$

Figure Source: Sergey Levine : UCB

Deterministic Policy Gradient Algorithm : Key Ideas

- Thus far, considered the policy function $\pi(\cdot|s)$ as a probability distribution over actions space and thus considered stochastic policies
- **Deterministic policy gradient algorithms (DPG)** instead models the policy as a deterministic decision : $a = \pi(s)$
- Specifically DDPG, an off-policy actor-critic algoirthm, can be thought of as DQN for continuous action space setting
- Interleaves between learning optimal action-value function $Q^*(s, a)$ and learning optimal policy $\pi^*(s)$
- Uses Bellman equation to learn $Q^*(s, a)$ and policy gradients to learn $\pi^*(s)$

Deterministic Policy Gradient Algorithm : Key Ideas

- Bellman equation is the starting point for learning optimal action-value function $Q^*(s, a)$.
- Optimal action-value function int the DQN setting is learnt using the following MSBE function

$$L_i(\phi_i) = \left[\mathbb{E}_{(s,a,r,s') \in D} \left(Q_{\phi_i}(s, a) - \underbrace{r + \max_{a'} Q_{\phi_i'}(s', a')}_{\text{target}} \right) \right]^2$$

- However, in the DDPG setting, we calculate the max over actions using the policy netwoek as follows,

$$L_i(\phi_i) = \left[\mathbb{E}_{(s,a,r,s') \in D} \left(Q_{\phi_i}(s, a) - \underbrace{r + Q_{\phi_i'}(s', \pi_\theta(s'))}_{\text{target}} \right) \right]^2$$

Deterministic Policy Gradient Algorithm : Key Ideas

- Policy is learnt by recoginizing that we are looking for a deterministic policy $\pi_\theta(s)$ that gives an action that maximizes $Q_\phi(s,a)$. Achieved by, performing gradient ascent on the following objective function

$$\max_\theta \mathbb{E}_{s \in \mathcal{D}} Q_\phi(s, \pi_\theta(s))$$

- Because the policy that is being learnt is deterministic, to make DDPG policies explore better, we add noise to their actions at training time.
  - OU noise
  - zero-mean Gaussian noise
- Target networks are updated using Polyak averaging
- The idea of deterministic policy gradient has connections to the stochastic policy gradient setting (in the limiting case)

Deep Deterministic Policy Gradient (DDPG)

Algorithm Deep Deterministic Policy Gradient

Initialize state s, critic ϕ, actor θ and replay buffer
Initialize target critic $\phi' \leftarrow \phi$, target actor $\theta' \leftarrow \theta$
for Repeat over several episodes **do**
    Initialize a random process N for exploration (eg. Ornstein-Uhlenbeck process), and observe initial state s
    for Repeat over transitions **do**
        Apply action $a = \pi_\theta(s) + N_t$, observe reward r and next state s', and store the transition (s, a, r, s') in the replay buffer
        Sample a random minibatch of transitions (s_i, a_i, r_i, s_i') from the buffer
        Compute SARSA target values $y_i = r_i + Q_{\phi'}(s_i', \pi_{\theta'}(s_i'))$
        Update critic by minimizing MSE loss $\frac{1}{n}\sum_i (y_i - Q_\phi(s_i, a_i))^2$
        Update actor using sampled deterministic policy gradient $\frac{1}{n}\sum_i \nabla_a Q_\phi(s_i, \pi_\theta(s_i)) \nabla_\theta \pi_\theta(s_i)$
        Perform soft updates on target networks
            $\phi' \leftarrow \tau\phi + (1-\tau)\phi'$
            $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$
    end for
end for

Double DQN

Maximization Bias

- Consider single-state MDP with 2 actions.
- Both actions have zero mean rewards (the agent does not know this information)
- Let $\hat{Q}(\cdot, a_1)$ and $\hat{Q}(\cdot, a_2)$ be (unbiased) finite sample estimates of Q for action $a_!$ and a_2 respectively
- The agent will prefer the action which has maximum $\hat{Q}$ based on sample estimates, although both actions have same expected mean reward

Maximization Bias

- Consider single-state MDP with 2 actions.
- One action has $-\epsilon$ (ϵ positive and small) mean reward and the other action has zero mean reward.
- Let $\hat{Q}(\cdot, a_1)$ and $\hat{Q}(\cdot, a_2)$ be (unbiased) finite sample estimates of Q for action a_1 and a_2 respectively
- In this case, the agent might choose action a_1 , depending on how the maximum $\hat{Q}$ values that is based on sample estimates show up, although action a_2 is clearly better in expectation

Extending the analogy, the (tabular or deep) Q-learning algorithm may actiually pick the suboptimal action for target computation and this causes over estimation of Q-values.

Issues in Tabular Q-Learning Approach

- The problem with vanilla tabular Q-Learning is that the same samples are being used to decide which action is the best (highest expected reward), and the same samples are also being used to estimate that action-value
- Break up the Q-learning update rule as follows

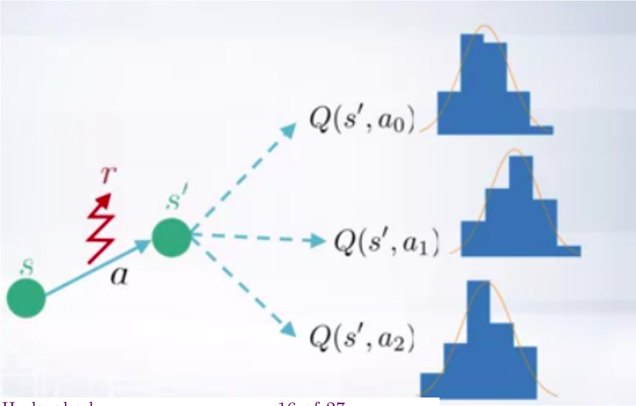
$$Q(s_t, a) \leftarrow Q(s_t, a) + \alpha \left(r_{t+1} + \gamma Q(s_{t+1}, \arg\max Q(s_{t+1}, a)) - Q(s_t, a) \right)$$

- If action value is overestimated, then it is chosen as the best action, and its overestimated value is used as the target

Slide Credit: Medium Blog : Ameet Deshpande

Persistence of the Problem in DQN

- For transition $(s_t, a_t, r_{t+1}, s_{t+1})$ the TD target of the Q-value update is $r_{t+1} + \gamma \arg\max_{a'} Q_\phi(s_{t+1}, a')$
- $Q_\phi(\cdot, \cdot)$ is a noisy estimate during training phase
- Therefore, $\max Q_\phi(\cdot, \cdot)$ would typically be overestimated during training



Evidence from Atari Games

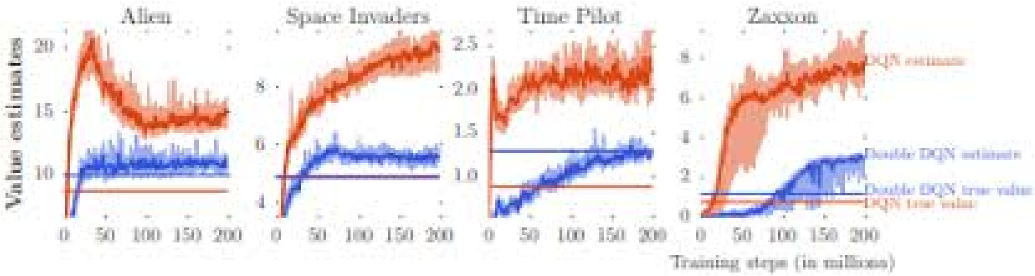


Figure Source: van Hasselt : DDQN paper

Possible Resolution ...

- There are two identical fair coins (we don't know they are fair)
- If a coin lands on head, we get one dollar; otherwise we lose a dollar
- Interested in answering the following questions
  - ★ Which coin will yield will more money in future flips ?
  - ★ How much can we expect to win or lose per flip using the coin from previous question ?
- Two ways to answer
  - ★ Flip each coin n few times and answer both questions
  - ★ Flip each coin n_1 times, answer the first question; collect fresh n_2 samples to answer second question based on the answer to the first question

The idea behind the second method is that we use separate samples to choose the best action and separate samples to use Q-values

Solution to Maximization Bias

- Have two different set of samples to decide the action and to evaluate the target
- The idea is to use two Q functions
- In the tabular Q-learning setting, for each transition quadruple $(s_t, a_t, r_{t+1}, s_{t+1})$ we flip a fair coin to decide any of the two update steps given below,

$$Q_1(s_t, a_t) \leftarrow Q_1(s_t, a_t) + \alpha \left(r_{t+1} + \gamma Q_2(s_{t+1}, \arg\max Q_1(s_{t+1}, a)) - Q_1(s_t, a_t) \right)$$

$$Q_2(s_t, a_t) \leftarrow Q_2(s_t, a_t) + \alpha \left(r_{t+1} + \gamma Q_1(s_{t+1}, \arg\max Q_2(s_{t+1}, a)) - Q_2(s_t, a_t) \right)$$

Double DQN

- In the DQN setting, we already have two Q networks (to address the moving target problem). So, we can take advantage of that by using the following update rule.
- In Double DQN, targets for the transition $(s_t, a_t, r_{t+1}, s_{t+1})$ is computed as follows,

$$Q^{original}(s_t, a_t) \leftarrow r + \gamma Q^{target}(s, \arg\max Q^{original}(s, a))$$

- In the above equation, the $\leftarrow$ actually means assigning targets which can then be picked up while sampling the replay buffer
- The fundamental idea is to use two Q_ϕ's (both are noisy estimates of true Q) in different ways so that the overestimation problem mellows down

Priortized Experience Replay

Priortized Experience Replay

- Replaying all transitions with equal probability is suboptimal
- Replay transitions in proportion to absolute Bellman error

$$\left| r + \gamma \max_{a'} Q_{\phi'}(s', a') - Q_{\phi}(s, a) \right|$$

- Leads to much faster learning

Priortized Experience Replay

- TD error for vanilla DQN is

$$\delta_i = r_t + \gamma \max_{a \in \mathcal{A}} Q_{\phi'}(s_{t+1}, a) - Q_\phi(s_t, a_t)$$

- TD error for DDQN is

$$\delta_i = r_t + \gamma Q_{\phi'}(s_{t+1}, \text{argmax}_{a \in \mathcal{A}} Q_\phi(s_{t+1}, a)) - Q_\phi(s_t, a_t)$$

- Priority for each entry in replay buffer D is given by $p_i = |\delta_i| + \epsilon$

Priortized Experience Replay

- Sample from replay buffer according to probability distribution

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

  with α determining the level of prioritization

- In order to compute the expectation

$$\min_\phi \mathbb{E}_{(s_t, a_t, r_t, s_{t+1}) \sim D} \left[\left(r_t + \gamma \max_{a \in \mathcal{A}} Q_{\phi'}(s_{t+1}, a) - Q_\phi(s_t, a_t) \right)^2 \right],$$

  it is essential to use the importance sampling weights in each mini-batch of the gradient update

$$w_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)} \right)^\beta$$

- The parameter β is an annealing term that is low (0.4 to 0.8) in the beginning of the training and tends to 1 towards the end of the training.
- The question on optimal choices of α and β during various phases of training depends on task at hand

Practical Tips

Practical Tips for DQN [2]

- DQN is more reliable on some tasks than others. Test your impementation on reliable tasks like Pong and Breakout: if it doesn't achieve good scores, something is wrong
- Large replay buffers improve robustness of DQN, and memory efficiency is important
- DQN converges slowly - for ATARI it is often necessary to wait for 10-40 million frames (couple of hours to a day of training on GPU) to see results significantly better than random policy. **Be Patient**
- Always run at least two different seeds when experimenting
- Learning rate scheduling is beneficial. Try high learning rates in initial exploration period

[2] Slide content from Schulman

Practical Tips for DQN [2]

- Try non-standard exploration schedules
- Do use Double DQN with priortized experience replay – significant improvement
- Use Huber loss on Bellman error

$$L_\delta(a) = \begin{cases} \frac{1}{2}a^2 & \text{for } |a| \le \delta, \\ \delta(|a| - \frac{1}{2}\delta), & \text{otherwise.} \end{cases}$$