Model Predictive Control and Reinforcement Learning
– Advanced Value-based Methods –

Joschka Boedecker and Moritz Diehl

University Freiburg

July 29, 2021

Lecture Overview

Acknowledgement

Slide contents are partially based on *Reinforcement Learning: An Introduction* by Sutton and Barto and the Reinforcement Learning lecture by David Silver.

Deep Deterministic Policy Gradient

- DDPG is an actor-critic method (*Continuous DQN*)
- Recall the DQN-target: $y_j = r_j + \gamma \max_a Q(s_{j+1}, a, \mathbf{w}^-)$
- In case of continuous actions, the maximization step is not trivial
- Therefore, we approximate deterministic actor μ representing the $\arg\max_a Q(s_{j+1}, a, \mathbf{w})$ by a neural network and update its parameters following the *Deterministic Policy Gradient Theorem*:
$$\nabla_\theta \leftarrow \frac{1}{N} \sum_j \nabla_a Q(s_j, a, \mathbf{w})|_{a=\mu(s_j)} \nabla_\theta \mu(s_j, \boldsymbol{\theta})$$
- Exploration by adding Gaussian noise to the output of μ

Deep Deterministic Policy Gradient

- The Q-function is fitted to the adapted TD-target:

$$y_j = r_j + \gamma Q(s_{j+1}, \mu(s_{j+1}, \boldsymbol{\theta}^-), \mathbf{w}^-)$$

- The parameters of target networks $\mu(\cdot, \boldsymbol{\theta}^-)$ and $Q(\cdot, \cdot, \mathbf{w}^-)$ are then adjusted with a soft update

$$\mathbf{w}^- \leftarrow (1-\tau)\mathbf{w}^- + \tau\mathbf{w} \text{ and } \boldsymbol{\theta}^- \leftarrow (1-\tau)\boldsymbol{\theta}^- + \tau\boldsymbol{\theta}$$

  with $\tau \in [0, 1]$
- DDPG is very popular and builds the basis for more SOTA actor-critic algorithms
- However, it can be quite unstable and sensitive to its hyperparameters

Deep Deterministic Policy Gradient

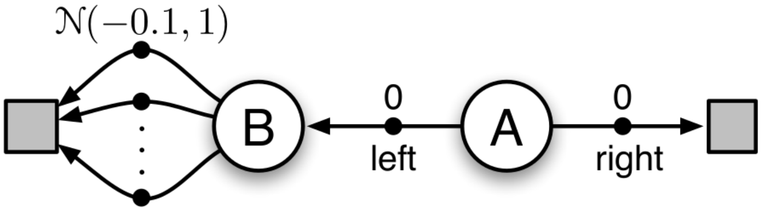
```
Initialize replay memory $D$ to capacity $N$
Initialize critic $Q$ and actor $\mu$ with random weights
for episode $i = 1, .., M$ do
    for $t = 1, .., T$ do
        select action $a_t = \mu(s_t, \boldsymbol{\theta}) + \epsilon$, where $\epsilon \sim \mathcal{N}(0, \sigma)$
        Store transition $(s_t, a_t, s_{t+1}, r_t)$ in $D$
        Sample minibatch of transitions $(s_j, a_j, r_j, s_{j+1})$ from D
        Set $y_j = \begin{cases} r_j & \text{if } s_{j+1} \text{ is terminal} \\ r_j + \gamma\, Q(s_{j+1}, \mu(s_{j+1}, \boldsymbol{\theta}^-), \mathbf{w}^-) & \text{else} \end{cases}$
        Update the parameters of $Q$ according to the TD-error
        Update the parameters of $\mu$ according to:

            $$\nabla_\theta \leftarrow \frac{1}{N} \sum_j \nabla_a Q(s_j, a, \mathbf{w})|_{a=\mu(s_j)} \nabla_\theta \mu(s_j, \boldsymbol{\theta})$$

        Adjust the parameters of the target networks via a soft update
    end
end
```

Overestimation Bias

- In all control algorithms so far, the target policy is created by the *maximization* of a value-function
- We thus consider the maximum over estimated values as an estimate of the maximum value
- This can lead to the so-called *overestimation bias*



Overestimation Bias

- Recall the Q-learning target: $R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$
- Imagine two random variables X_1 and X_2:

$$\mathbb{E}[\max(X_1, X_2)] \geq \max(\mathbb{E}[X_1], \mathbb{E}[X_2])$$

- $Q(S_{t+1}, a)$ is not perfect – it can be *noisy*:

$$\max_a Q(S_{t+1}, a) = \overbrace{Q(S_{t+1}, \underbrace{\arg\max_a Q(S_{t+1}, a)}_{\text{action comes from } Q})}^{\text{value comes from } Q}$$

- If the noise in these is decorrelated, the problem goes away!

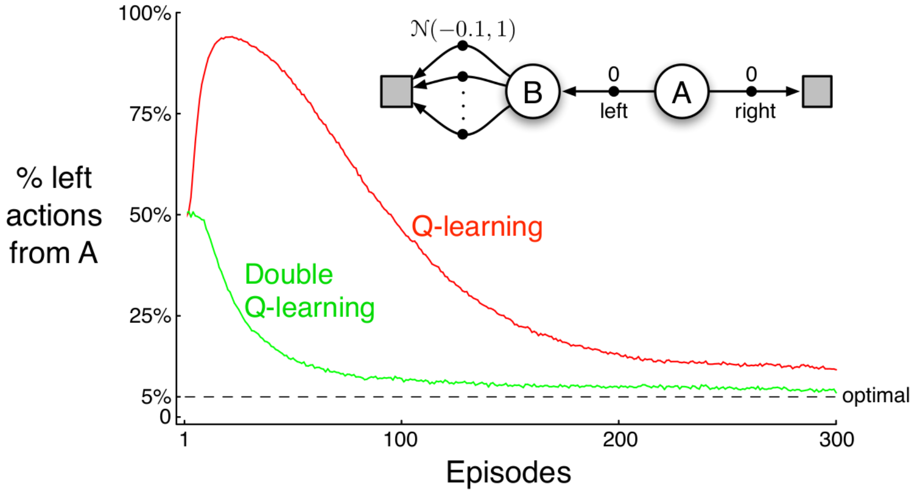
Double Q-learning

Double Q-learning, for estimating $Q_1 \approx Q_2 \approx q_*$

```
Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$
Initialize $Q_1(s,a)$ and $Q_2(s,a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, such that $Q(terminal, \cdot) = 0$

Loop for each episode:
   Initialize $S$
   Loop for each step of episode:
      Choose $A$ from $S$ using the policy $\varepsilon$-greedy in $Q_1 + Q_2$
      Take action $A$, observe $R$, $S'$
      With 0.5 probabilility:
         $Q_1(S,A) \leftarrow Q_1(S,A) + \alpha\Big(R + \gamma Q_2\big(S', \operatorname{argmax}_a Q_1(S',a)\big) - Q_1(S,A)\Big)$
      else:
         $Q_2(S,A) \leftarrow Q_2(S,A) + \alpha\Big(R + \gamma Q_1\big(S', \operatorname{argmax}_a Q_2(S',a)\big) - Q_2(S,A)\Big)$
      $S \leftarrow S'$
   until $S$ is terminal
```

Double Q-learning



TD3

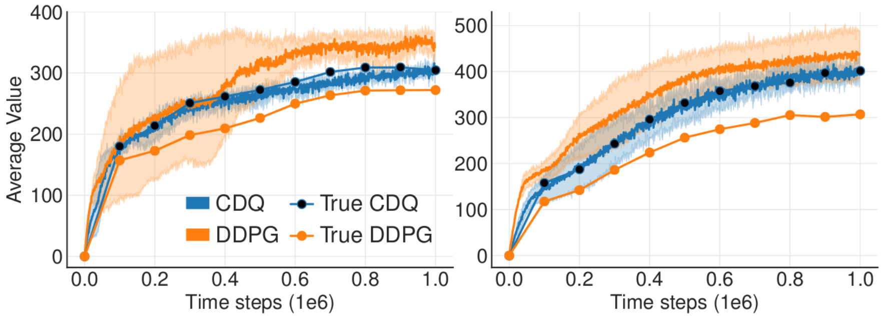
TD3 adds three adjustments to vanilla DDPG

- Clipped Double Q-Learning
- Delayed Policy Updates
- Target-policy smoothing

TD3: Clipped Double Q-Learning

- In order to alleviate the overestimation bias (which is also present in actor-critic methods), TD3 learns two approximations of the action-value function
- It the takes the minimum of both predictions as the second part of the TD-target:

$$y_j = r_j + \gamma \min_{i \in \{1,2\}} Q(s_{j+1}, \mu(s_{j+1}), \mathbf{w}_i^-)$$



TD3: Delayed Policy Updates

- Due to the mutual dependency between actor and critic updates...
  - values can diverge when the policy leads to overestimation and
  - the policy will lead to bad regions of the state-action space when the value estimates lack in (relative) accuracy
- Therefore, policy updates on states where the value-function has a high prediction error can cause divergent behaviour
- We already know how to compensate for that: target networks
- Freeze target and policy networks between d updates of the value function
- This is called a *Delayed Policy Update*

TD3: Target-policy Smoothing

- *Target-policy Smoothing* adds Gaussian noise to the next action in target calculation
- It transforms the Q-update towards an Expected SARSA update fitting the value of a small area around the target-action:

$$y_j = r_j + \gamma \min_{i \in \{1,2\}} Q(s_{j+1}, \mu(s_{j+1}) + \text{clip}(\epsilon, -c, c), \mathbf{w}_i^-),$$

where $\epsilon \sim \mathcal{N}(0, \sigma)$

TD3: Ablation

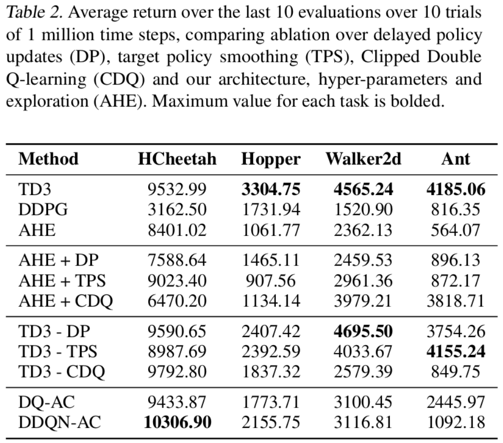


Table 2. Average return over the last 10 evaluations over 10 trials of 1 million time steps, comparing ablation over delayed policy updates (DP), target policy smoothing (TPS), Clipped Double Q-learning (CDQ) and our architecture, hyper-parameters and exploration (AHE). Maximum value for each task is bolded.

Method	HCheetah	Hopper	Walker2d	Ant
TD3	9532.99	**3304.75**	**4565.24**	**4185.06**
DDPG	3162.50	1731.94	1520.90	816.35
AHE	8401.02	1061.77	2362.13	564.07
AHE + DP	7588.64	1465.11	2459.53	896.13
AHE + TPS	9023.40	907.56	2961.36	872.17
AHE + CDQ	6470.20	1134.14	3979.21	3818.71
TD3 - DP	9590.65	2407.42	**4695.50**	3754.26
TD3 - TPS	8987.69	2392.59	4033.67	**4155.24**
TD3 - CDQ	9792.80	1837.32	2579.39	849.75
DQ-AC	9433.87	1773.71	3100.45	2445.97
DDQN-AC	**10306.90**	2155.75	3116.81	1092.18

Soft Actor-Critic

- Soft Actor-Critic: entropy-regularized value-learning
- The policy is trained to maximize a trade-off between expected return and entropy ($H(P) = \mathbb{E}_{x\sim P}[-\log P(x)]$), a measure of randomness in the policy:

$$\pi_* = \arg\max_{\pi} \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} R_{t+1} + \alpha H(\pi(\cdot|S_t = s_t)) \right]$$

- The value functions are then defined as:

$$v_\pi(s) = \mathbb{E}_\pi[\sum_{t=0}^{\infty} R_{t+1} + \alpha H(\pi(\cdot|S_t = s_t))|S_0 = s, A_0 = a]$$

and

$$q_\pi(s,a) = \mathbb{E}_\pi[\sum_{t=0}^{\infty} R_{t+1} + \alpha \sum_{t=1}^{\infty} \gamma^t H(\pi(\cdot|S_t = s_t))|S_0 = s, A_0 = a]$$

- And their relation as: $v_\pi(s) = \mathbb{E}_\pi[q_\pi(s,a)] + \alpha H(\pi(\cdot|S_t = s))$

Soft Actor-Critic

- The corresponding Bellman equation for q_π is

$$\begin{aligned} q_\pi(s_t, a_t) &= \mathbb{E}_{\pi,p}[R_{t+1} + \gamma(q_\pi(s_{t+1}, a_{t+1}) + \alpha H(\pi(\cdot|S_{t+1} = s_{t+1}))] \\ &= \mathbb{E}_{\pi,p}[R_{t+1} + \gamma v_\pi(s_{t+1})] \end{aligned}$$

- There are initial results that *Maximum Entropy RL*-methods lead to more composable policies

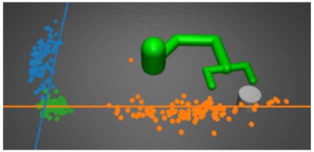


Fig. 2. Two independent policies are trained to push the cylinder to the orange line and blue line, respectively. The colored circles show samples of the final location of the cylinder for the respective policies. When the policies are combined, the resulting policy learns to push the cylinder to the lower intersection of the lines (green circle indicates final location). No additional samples from the environment are used to train the combined policy. The combined policy learns to satisfy both original goals, rather than simply averaging the final cylinder location.

Further ressources

If you want to get an even more detailed overview about the current SOTA, you can have a look at OpenAI SpinningUp:

`https://spinningup.openai.com/en/latest/index.html`