Lecture 11: Advanced Value-based Methods

Friday, January 28, 2022

Reinforcement Learning, Winter Term 2021/22

Joschka Boedecker, Gabriel Kalweit, Jasper Hoffmann

Neurorobotics Lab
University of Freiburg

Lecture Overview

1. Recap
2. Prioritized Experience Replay
3. Distributional RL
4. DDPG
5. TD3
6. Soft Actor-Critic
7. Wrapup

Lecture Overview

Recap: Policy Gradient Methods

- *Policy Gradient Methods* consider a parameterized *policy*

$$\pi(a|s, \boldsymbol{\theta}) = \Pr\{A_t = a | S_t = s, \boldsymbol{\theta}_t = \boldsymbol{\theta}\},$$

  with parameters $\boldsymbol{\theta}$
- We update these parameters based on the gradient of some performance measure $J(\boldsymbol{\theta})$ (expected return) that we want to maximize, i.e. via *gradient ascent*:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \widehat{\nabla J(\boldsymbol{\theta}_t)},$$

  where $\widehat{\nabla J(\boldsymbol{\theta}_t)} \in \mathbb{R}^d$ is a stochastic estimate whose expectation approximates the gradient of the performance measure w.r.t. $\boldsymbol{\theta}_t$

Policy Gradient Theorem

For any differentiable policy $\pi(a|s, \boldsymbol{\theta})$ and any of the introduced policy objective functions, the policy gradient is:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_\pi[\nabla_{\boldsymbol{\theta}} \log \pi(a|s, \boldsymbol{\theta}) q_\pi(s, a)]$$

Recap: Deep Q-Networks (DQN)

Algorithm 1 DQN

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights
for *episode* $i = 1, .., M$ **do**
    for $t = 1, .., T$ **do**
        select action a_t ϵ-greedily
        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 $y_j = \begin{cases} r_j & \text{if } s_{j+1} \text{ is terminal} \\ r_j + \gamma \max_{a'} Q(s_{j+1}, a', \mathbf{w}^-) & \text{else} \end{cases}$
        Update the parameters of Q according to MSE between y_j and $Q(s_j, a_j, \mathbf{w})$
        Update target network

Lecture Overview

Prioritized Experience Replay

Recall DQN with experience replay:

- Take action a_t according to ϵ-greedy policy
- Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory D
- Sample random mini-batch of transitions (s, a, r, s') from D
- Optimize MSE between Q-network and Q-learning targets, e.g.

$$L(\mathbf{w}) = \mathbb{E}_{s,a,r,s' \sim D}\left[(r + \gamma \max_{a'} Q(s', a', \mathbf{w}) - Q(s, a, \mathbf{w}))^2\right]$$

Problem: In many tasks, the number of "unimportant" samples grows faster than the number of "interesting" transitions, which guide the agent towards the goal (e.g. in sparse reward tasks).
$\Rightarrow$ Sampling mini-batches uniformly can be slow and inefficient!

Prioritized Experience Replay

Prioritized Experience Replay:

- Take action a_t according to ϵ-greedy policy, obtain $(s, a, r, s')_i$
- Measure the "importance" or *priority* p_i of a transition $(s, a, r, s')_i$ by the magnitude of the td-error

$$p_i \leftarrow |r + \gamma \max_{a'} Q(s', a', \mathbf{w}) - Q(s, a, \mathbf{w})|$$

- Sample transitions with probability relative to their importance, e.g.

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}, \quad \alpha > 0$$

  instead of taking only the samples with highest p_i to avoid sampling only noisy td-samples and forgetting samples with small priority.
- Can be implemented efficiently in a priority queue using a binary heap ($O(1)$ sampling, $O(log(n))$ for updating priorities)

Lecture Overview

Distributional RL

- Standard RL is only concerned with the *expected* long-term value of a transition, e.g. recall the bellman equation

$$Q^\pi(s,a) = \mathbb{E}_{s' \sim p(\cdot|s,a),\, a' \sim \pi(s')}[r(s,a) + Q^\pi(s',a')]$$

- The Q-function averages over two sources of randomness (i.e. random variables):
  - The transition $p(s'|s,a)$
  - The reward function $r(s,a)$
- In distributional RL, we look at the *distribution* over the long-term value of a transition

$$Z^\pi(s,a) = R(s,a) + \sum_{t=1}^{\infty} \gamma^t R_t,$$

  where $Z^\pi(s,a)$ is called the *value distribution*.
- Relationship to Q-function: $Q^\pi(s,a) = \mathbb{E}[Z^\pi(s,a)]$

Distributional RL

- We can define a transition operator P with $PZ^\pi(s,a) := Z^\pi(S',A')$, i.e the value distribution that depends on the (random) next state and action $S' \sim p(\cdot|s,a)$, $A' \sim \pi(\cdot|S')$.
- We can define the Bellman operator $\mathcal{T}^\pi Z^\pi(s,a)$ as:

$$\mathcal{T}^\pi Z^\pi(s,a) := R(s,a) + \gamma P^\pi Z^\pi(s,a)$$

  and a corresponding distributional Bellman optimality equation exists (similar to the Q-learning update).
- Convergence to an optimal value distribution Z^* is not guaranteed
- A sample-based algorithm similar to DQN can be derived that shows improved performance on most Atari tasks.
- Can model multi-modalities in value distributions and may help with non-stationarities leading to more stable learning.

Distributional RL

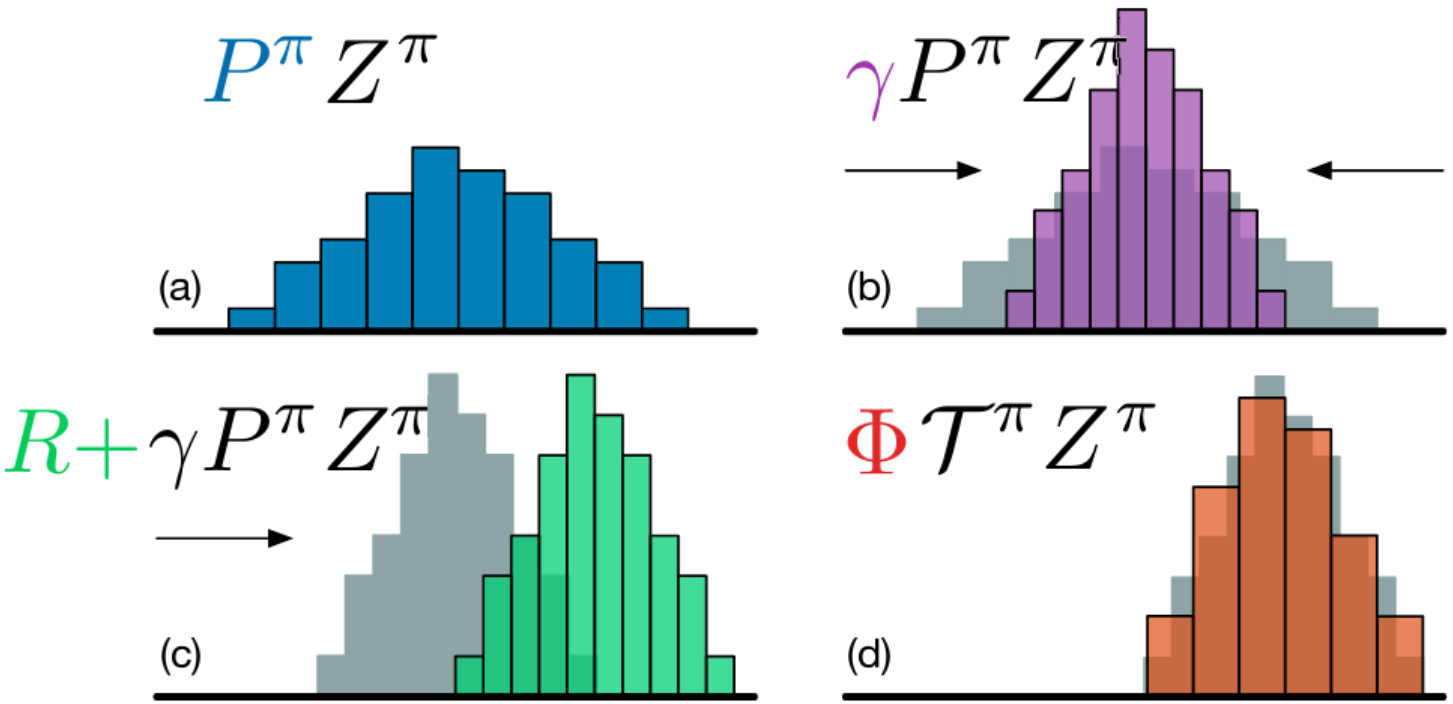


Figure 1. A distributional Bellman operator with a deterministic reward function: (a) Next state distribution under policy π, (b) Discounting shrinks the distribution towards 0, (c) The reward shifts it, and (d) Projection step (Section 4).

Lecture Overview

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

Algorithm 2 DDPG

Initialize replay memory D to capacity N
Initialize critic Q and actor μ 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 μ 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

Lecture Overview

TD3

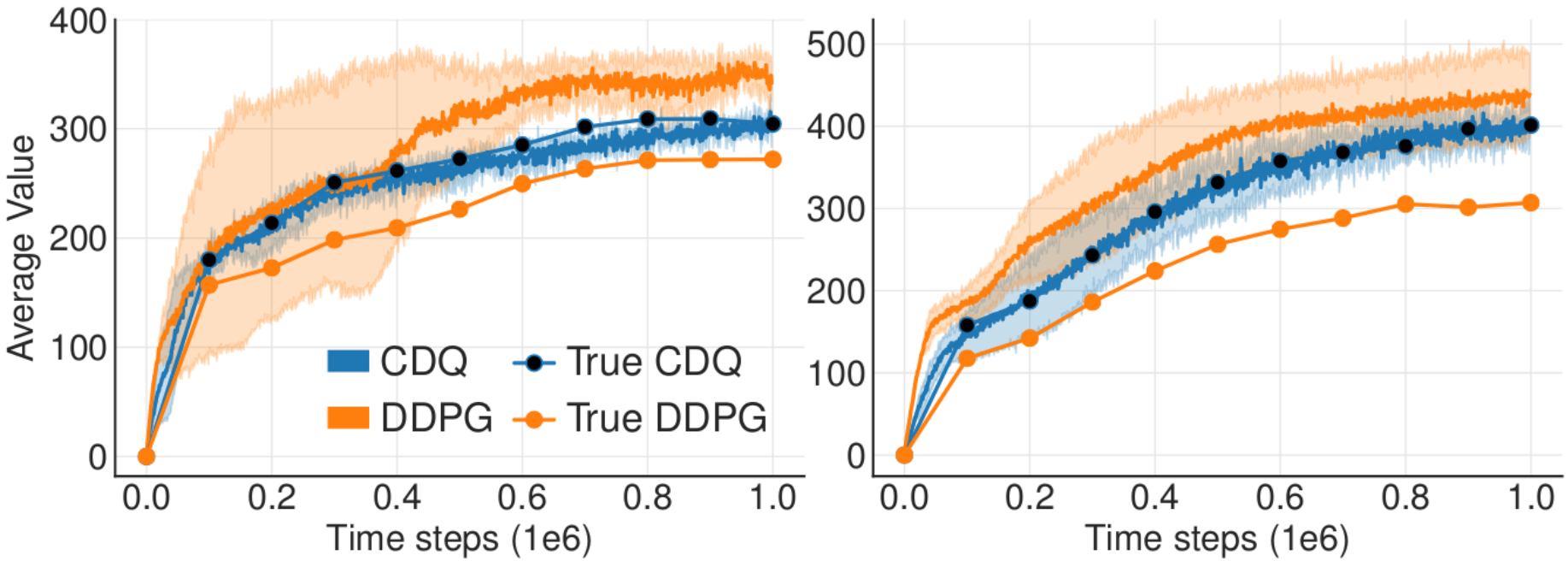
TD3 adds three adjustments to vanilla DDPG

- 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:

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

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

Lecture Overview

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

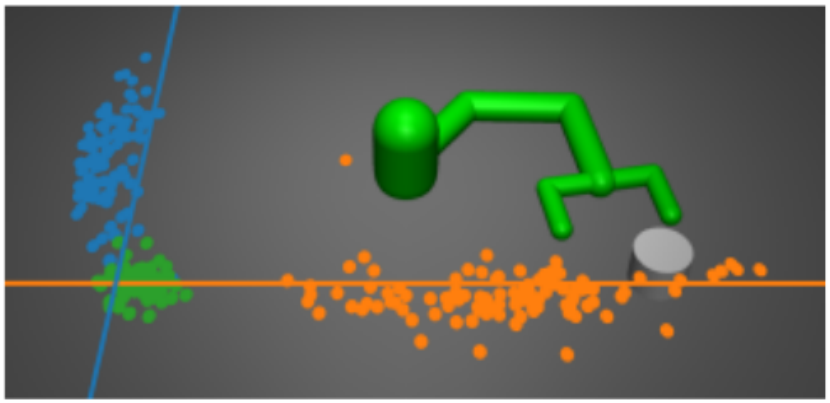


Fig. 2. Two independent policies are trained to push the cylinder to the orange line and blue line, respectively. The colored circles show samples of the final location of the cylinder for the respective policies. When the policies are combined, the resulting policy learns to push the cylinder to the 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.

Lecture Overview

Summary by Learning Goals

Having heard this lecture, you can now...

- apply more sophisticated value-function methods to problems of higher complexity
- think of ways to improve deep value-function based methods
- provide an overview about the current SOTA

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`