Lecture 8: Off-policy Methods with Function Approximation

Friday, December 17, 2021

Reinforcement Learning, Winter Term 2021/22

Joschka Boedecker, Gabriel Kalweit, Jasper Hoffmann

Neurorobotics Lab
University of Freiburg

Lecture Overview

Lecture Overview

Recap: Function Approximation in Reinforcement Learning

We want to update the weights w.r.t. the *Mean Squared Value Error* of the prediction:

$$\begin{aligned}\mathbf{w} &\leftarrow \mathbf{w} - \frac{1}{2}\alpha\nabla[v_\pi(S_t) - \hat{v}(S_t, \mathbf{w})]^2 \\ &\leftarrow \mathbf{w} + \alpha[v_\pi(S_t) - \hat{v}(S_t, \mathbf{w})]\nabla\hat{v}(S_t, \mathbf{w})\end{aligned}$$

However, we don't have $v_\pi(S_t)$.

Recap: Function Approximation in Reinforcement Learning

Gradient MC

$\mathbf{w} \leftarrow \mathbf{w} + \alpha[G_t - \hat{v}(S_t, \mathbf{w})]\nabla\hat{v}(S_t, \mathbf{w})$

Semi-gradient TD(0)

$\mathbf{w} \leftarrow \mathbf{w} + \alpha[R_{t+1} + \gamma\hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w})]\nabla\hat{v}(S_t, \mathbf{w})$

Why are bootstrapping methods, defined this way, called *semi-gradient methods*? They take into account the effects of changing $\mathbf{w}$ w.r.t. the prediction, but not w.r.t. the target!

Recap: Semi-gradient SARSA

Episodic Semi-gradient Sarsa for Estimating $\hat{q} \approx q_*$

Input: a differentiable action-value function parameterization $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$
Algorithm parameters: step size $\alpha > 0$, small $\varepsilon > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:
    $S, A \leftarrow$ initial state and action of episode (e.g., ε-greedy)
    Loop for each step of episode:
        Take action A, observe R, S'
        If S' is terminal:
            $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$
            Go to next episode
        Choose A' as a function of $\hat{q}(S', \cdot, \mathbf{w})$ (e.g., ε-greedy)
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \gamma\hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$
        $S \leftarrow S'$
        $A \leftarrow A'$

Lecture Overview

Off-policy Learning

- We want to learn the optimal policy, but we have to account for the problem of *maintaining exploration*
- We call the (optimal) policy to be learned the *target policy* π and the policy used to generate behaviour the *behaviour policy* b
- We say that learning is from data *off* the target policy – thus, those methods are referred to as *off-policy learning*
- Today: Off-policy learning methods with function approximation

Semi-gradient Off-policy TD(0)

Replace the update to an array to an update to weight vector $\mathbf{w}$.

Recap: Importance Sampling Ratio

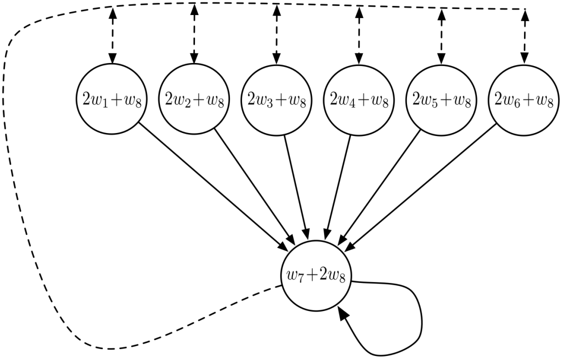
$$\rho_t = \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$$

Semi-gradient Off-policy TD(0)

$\mathbf{w} \leftarrow \mathbf{w} + \alpha \rho_t \delta_t \nabla \hat{v}(S_t, \mathbf{w})$
where $\delta_t = R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w})$

Lecture Overview

Baird's Counterexample



$\pi(\text{solid}|\cdot) = 1$

$b(\text{dashed}|\cdot) = 6/7$

$b(\text{solid}|\cdot) = 1/7$

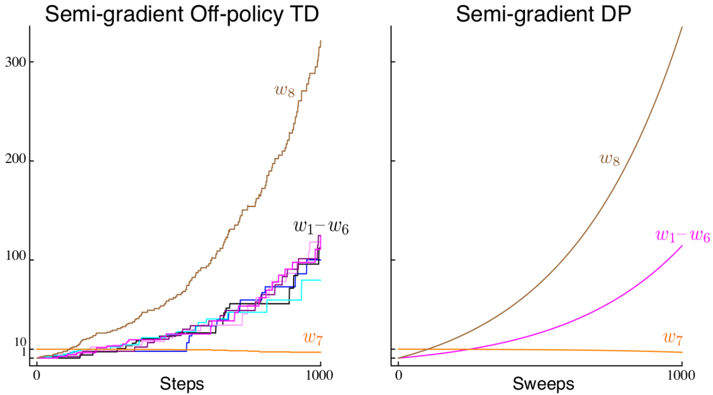
$\gamma = 0.99$

The reward is 0 for all transitions, hence $v_\pi(s) = 0$. This could be exactly approximated by $\mathbf{w} = \mathbf{0}$.

Baird's Counterexample

Semi-gradient DP

$$\mathbf{w} \leftarrow \mathbf{w} + \frac{\alpha}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} (\mathbb{E}[R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}) | S_t = s] - \hat{v}(s, \mathbf{w})) \nabla \hat{v}(s, \mathbf{w})$$



The Deadly Triad

The combination of

- Function Approximation,
- Bootstrapping and
- Off-policy Learning

is known as the *Deadly Triad*, since it can lead to stability issues and divergence.

Lecture Overview

Gradient-TD Methods

To this point, the TD-methods discussed did not leverage the true gradient (they are called semi-gradient methods). Recall the Bellman Equation:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')]$$

Bellman Error

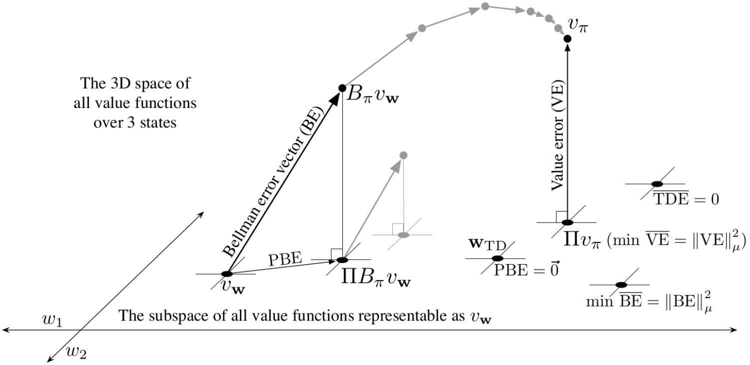
$$\begin{aligned}\bar{\delta}_{\mathbf{w}}(s) &= \left(\sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_{\mathbf{w}}(s')]\right) - v_{\mathbf{w}}(s) \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma v_{\mathbf{w}}(S_{t+1}) - v_{\mathbf{w}}(S_t)|S_t = s, A_t \sim \pi]\end{aligned}$$

Gradient-TD Methods

Projected Bellman Error

The Projected Bellman Error is the projection of the Bellman Error back into the representable space:

$$\overline{\mathsf{PBE}} = \|\Pi\bar{\delta}_{\mathbf{w}}\|_{\mu}^{2}$$



Gradient-TD Methods

Projection Matrix for linear FA

The projection matrix for linear FA can be represented as an $|\mathcal{S}| \times |\mathcal{S}|$ matrix:

$$\Pi = \mathbf{X}(\mathbf{X}^\top \mathbf{D}\mathbf{X})^{-1}\mathbf{X}^\top \mathbf{D}$$

$$\begin{aligned}
\overline{\mathsf{PBE}}(\mathbf{w}) &= \|\Pi\bar{\delta}_{\mathbf{w}}\|_\mu^2 \\
&= (\Pi\bar{\delta}_{\mathbf{w}})^\top \mathbf{D}\Pi\bar{\delta}_{\mathbf{w}} \\
&= \bar{\delta}_{\mathbf{w}}^\top \Pi^\top \mathbf{D}\Pi\bar{\delta}_{\mathbf{w}} \\
&= \bar{\delta}_{\mathbf{w}}^\top \mathbf{D}\mathbf{X}(\mathbf{X}^\top \mathbf{D}\mathbf{X})^{-1}\mathbf{X}^\top \mathbf{D}\bar{\delta}_{\mathbf{w}} \\
&= (\mathbf{X}^\top \mathbf{D}\bar{\delta}_{\mathbf{w}})^\top (\mathbf{X}^\top \mathbf{D}\mathbf{X})^{-1}(\mathbf{X}^\top \mathbf{D}\bar{\delta}_{\mathbf{w}})
\end{aligned}$$

The gradient with respect to $\mathbf{w}$ is:

$$\nabla\overline{\mathsf{PBE}}(\mathbf{w}) = 2\nabla[\mathbf{X}^\top \mathbf{D}\bar{\delta}_{\mathbf{w}}]^\top (\mathbf{X}^\top \mathbf{D}\mathbf{X})^{-1}(\mathbf{X}^\top \mathbf{D}\bar{\delta}_{\mathbf{w}})$$

Gradient-TD Methods

Assume μ to be the distribution of state visited under the behaviour policy.

1. $$\mathbf{X}^\top \mathbf{D} \bar{\delta}_{\mathbf{w}} = \sum_s \mu(s) \mathbf{x}(s) \bar{\delta}_{\mathbf{w}}(s) = \mathbb{E}[\rho_t \delta_t \mathbf{x}_t]$$

2. $$\begin{aligned} \nabla [\mathbf{X}^\top \mathbf{D} \bar{\delta}_{\mathbf{w}}]^\top &= \mathbb{E}[\rho_t \nabla \delta_t^\top \mathbf{x}_t^\top] \\ &= \mathbb{E}[\rho_t \nabla (R_{t+1} + \gamma \mathbf{w}^\top \mathbf{x}_{t+1} - \mathbf{w}^\top \mathbf{x}_t)^\top \mathbf{x}_t^\top] \\ &= \mathbb{E}[\rho_t (\gamma \mathbf{x}_{t+1} - \mathbf{x}_t) \mathbf{x}_t^\top] \end{aligned}$$

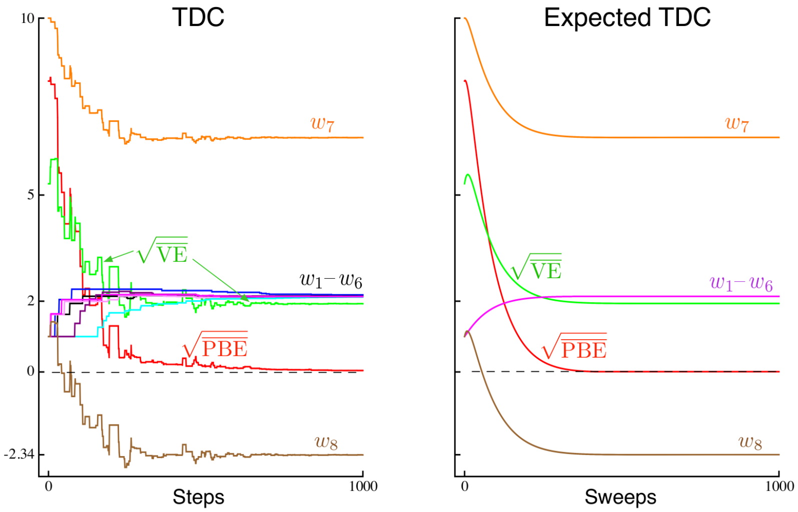
3. $$\mathbf{X}^\top \mathbf{D} \mathbf{X} = \sum_s \mu(s) \mathbf{x}_s \mathbf{x}_s^\top = \mathbb{E}[\mathbf{x}_t \mathbf{x}_t^\top]$$

Gradient-TD Methods

- We can thus rewrite the gradient as:

$$\nabla \overline{\mathrm{PBE}}(\mathbf{w}) = 2\mathbb{E}[\rho_t(\gamma \mathbf{x}_{t+1} - \mathbf{x}_t)\mathbf{x}_t^\top]\mathbb{E}[\mathbf{x}_t\mathbf{x}_t^\top]\mathbb{E}[\rho_t\delta_t\mathbf{x}_t]$$

- Store the last two factors in d-vector $\mathbf{v} \approx \mathbb{E}[\mathbf{x}_t\mathbf{x}_t^\top]^{-1}\mathbb{E}[\rho_t\delta_t\mathbf{x}_t]$ and update via $\mathbf{v} \leftarrow \mathbf{v} + \beta\rho_t(\delta_t - \mathbf{v}^\top\mathbf{x}_t)\mathbf{x}_t$
- Estimate the first part via sampling
- This method and variants of it are called Gradient-TD methods


TDC
Expected TDC
10
5
2
0
-2.34
w_7
$\sqrt{\overline{\text{VE}}}$
$w_1 - w_6$
$\sqrt{\overline{\text{PBE}}}$
w_8
0
1000
Steps
Sweeps

Lecture Overview

Neural Fitted-Q Iteration (NFQ) [Riedmiller 2005]

- Model-free off-policy RL algorithm that works on continuous state and discrete action spaces
- Q-function is represented by a multi-layer perceptron
- One of the first approaches that combined RL with ANNs, predecessor of DQN

Neural Fitted-Q Iteration (NFQ) [Riedmiller 2005]

Algorithm 1 NFQ

for *iteration* $i = 1, .., N$ **do**

  sample trajectory with ϵ-greedy exploration and add to memory D

  initialize network weights randomly

  generate pattern set $P = \{(x_j, y_j) | j = 1..|D|\}$ with

  $x_j = (s_j, a_j)$ and $y_j = \begin{cases} r_j & \text{if } s_j \text{ is terminal} \\ r_j + \gamma \max_{a'} Q(s_{j+1}, a', \mathbf{w}_i) & \text{else} \end{cases}$

  for *iteration* $k = 1, .., K$ **do**

    Fit weights according to:

$$L(\mathbf{w}_i) = \frac{1}{|D|} \sum_{j=1}^{|D|} (y_j - Q(x_j, \mathbf{w}_i))^2$$

Deep Q-Networks (DQN)

DQN provides a stable solution to deep RL:

- Use experience replay (as in NFQ)
- Sample minibatches (as opposed to Full Batch in NFQ)
- Freeze target Q-networks (no target networks in NFQ)
- Optional: Clip rewards or normalize network adaptively to sensible range

Deep Q-Networks: Experience Replay

To remove correlations, build data set from agent's own experience

- 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.

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

Deep Q-Networks: Target Networks

To avoid oscillations, fix parameters used in Q-learning target

- Compute Q-learning targets w.r.t. old, fixed parameters $\mathbf{w}^-$

$$r + \gamma \arg\max_{a'} Q(s', a', \mathbf{w}^-)$$

- Optimize MSE between Q-network and Q-learning targets

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

- Periodically update fixed parameters $\mathbf{w}^- \leftarrow \mathbf{w}$
    - hard update: update target network every N steps
    - slow update: slowly update weights of target network every step by

$$\mathbf{w}^- \leftarrow (1 - \tau)\mathbf{w}^- + \tau \mathbf{w}$$

Deep Q-Networks (DQN)

Algorithm 2 DQN

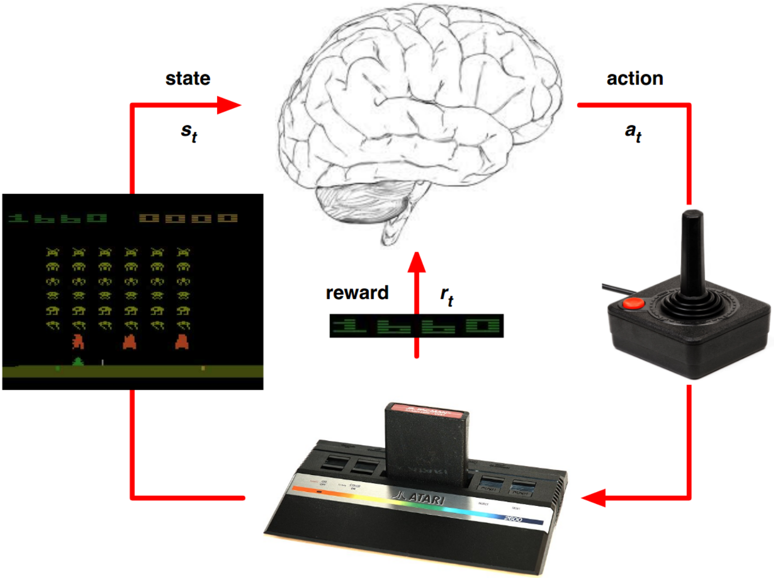
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights
for *episode* $i = 1, .., M$ **do**
  for $t = 1, .., T$ **do**
    select action a_t ϵ-greedily
    Store transition (s_t, a_t, s_{t+1}, r_t) in D
    Sample minibatch of transitions (s_j, a_j, r_j, s_{j+1}) 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:

$$\nabla \mathbf{w}_i L_i(\mathbf{w}_i) = \mathbb{E}_{s,a,s,r\sim D}[(r + \gamma \max_{a'} Q(s', a', \mathbf{w}_i) - Q(s, a, \mathbf{w}_i))\nabla_{\mathbf{w}_i} Q(s, a, \mathbf{w}_i)]$$

    Update target network

This is your exercise

Deep Q-Networks: Reinforcement Learning in Atari



Deep Q-Networks: Reinforcement Learning in Atari

- End-to-end learning of values $Q(s, a)$ from pixels s
- Input state s is a stack of raw pixels from the last 4 frames
- Output is $Q(s, a)$ for 18 joystick/button positions
- Reward is change in score for that step

How much does DQN help?

	Q-Learning	Q-Learning + Target Q	Q-Learning + Replay	DQN Q-learning + Replay + Target Q
Breakout	3	10	241	**317**
Enduro	29	142	831	**1006**
River Raid	1453	2868	4103	**7447**
Seaquest	276	1003	831	**2894**
Space Invaders	302	373	826	**1089**

Lecture Overview

Summary by Learning Goals

Having heard this lecture, you can now...

- Explain the difficulties that may arise in off-policy learning with FA
- Apply deep Q-learning and explain why we include replay memory and target networks