Model Predictive Control and Reinforcement Learning
– Off-Policy Control with Function Approximation –

Joschka Boedecker and Moritz Diehl

University Freiburg

July 16, 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.

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*

Importance Sampling

- Weight returns according to the relative probability of target and behaviour policy
- Define state-transition probabilities $p(s'|s,a)$ as
  $p(s'|s,a) = \text{Pr}\{S_t = s'|S_{t-1} = s, A_{t-1} = a\} = \sum_{r\in\mathcal{R}} p(s', r|s,a)$
- The probability of the subsequent trajectory under any policy π, starting in S_t, then is:

$$\begin{aligned}
&\text{Pr}\{A_t, S_{t+1}, A_{t+1}, \ldots S_T | S_t, A_{t:T-1} \sim \pi\} \\
&= \pi(A_t|S_t)p(S_{t+1}|S_t, A_t)\pi(A_{t+1}|S_{t+1}) \cdots p(S_T|S_{T-1}, A_{T-1}) \\
&= \prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k)
\end{aligned}$$

Importance Sampling

The relative probability therefore is:

Definition: Importance Sampling Ratio

$$\rho_{t:T-1} = \frac{\prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k)}{\prod_{k=t}^{T-1} b(A_k|S_k)p(S_{k+1}|S_k, A_k)} = \frac{\prod_{k=t}^{T-1} \pi(A_k|S_k)}{\prod_{k=t}^{T-1} b(A_k|S_k)}$$

The expectation of the returns by b is $\mathbb{E}[G_t|S_t = s] = v_b(s)$. However, we want to estimate the expectation under π. Given the importance sampling ratio, we can transform the MC returns by b to yield the expectation under π:

$$\mathbb{E}[\rho_{t:T-1}G_t|S_t = s] = v_\pi(s).$$

Importance Sampling can come with a vast increase in variance.

Off-policy MC Prediction and Semi-gradient TD(0)

To use importance sampling with function approximation, replace the update to an array to an update to weight vector $\mathbf{w}$, and correct it with the importance sampling weight.

Off-policy MC Prediction

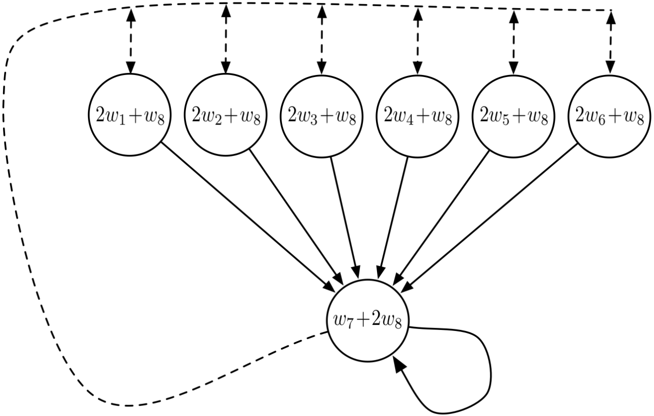
$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \rho_{t:T-1} [G_t - \hat{v}(S_t, \mathbf{w})] \nabla \hat{v}(S_t, \mathbf{w})$$

Semi-gradient Off-policy TD(0)

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \rho_t \delta_t \nabla \hat{v}(S_t, \mathbf{w})$$

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

Baird's Counterexample



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

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

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

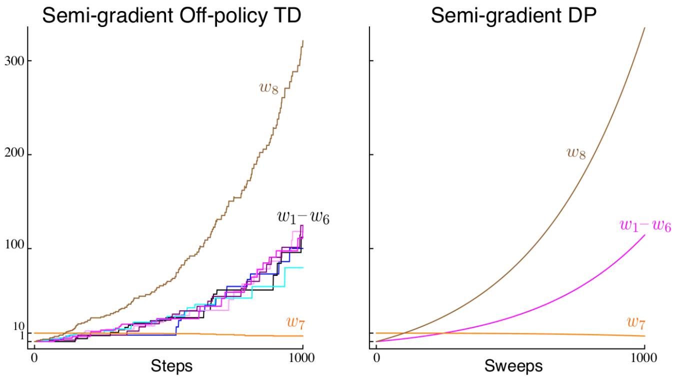
$\gamma = 0.99$

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.

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]

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

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

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

Fit weights according to:

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

end

end

Algorithm 1: NFQ

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

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

- 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

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

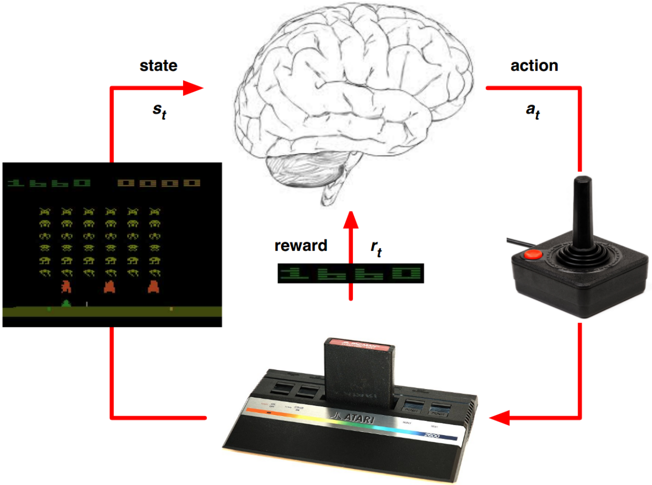
Deep Q-Networks (DQN)

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
  end
end

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**