Model Predictive Control and Reinforcement Learning
– TD Methods and Function Approximation –

Joschka Boedecker and Moritz Diehl

University of Freiburg

Fall School on Model Predictive Control and Reinforcement Learning
Freiburg, 6-10 October 2025

universität freiburg

NTNU | Norwegian University of Science and Technology

Recap: Dynamic Programming

Last lecture: Planning by dynamic programming, solve a *known*, *discrete* MDP.

Policy Iteration

Alternate **evaluating** the value function v_π and **improving** the policy π to convergence.

$$\pi_0 \xrightarrow{E} V^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V^{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \cdots \xrightarrow{I} \pi^* \xrightarrow{E} V^*$$

Value Iteration

Evaluate just once and combine it with the policy improvement step.

$$\begin{aligned} V_{k+1}(s) &\doteq \max_a \mathbb{E}\left[R_{t+1} + \gamma V_k(S_{t+1}) | S_t = s, A_t = a\right] \\ &= \max_a \sum_{s'} P(s'|s,a)\left[r(s,a) + \gamma V_k(s')\right] \end{aligned}$$

Approximate Dynamic Programming

Dynamic programming is optimal, but it requires knowledge of the dynamics and is too computationally expensive $\rightarrow$ Approximate Dynamic Programming

$$V_{k+1}(s) = \sum_{s',r} P(s'|s,a)\left[r(s,a) + \gamma V_k(s')\right]$$

In this lecture:

- **Local state updates:** We update only the visited states s, s' instead of solving the entire system at once.
- **No model required:** The transition model $P(s'|s,a)$ (and the reward function $r(s,a)$) is not needed; we sample s', r directly from interaction.
- **Function approximation:** Instead of a tabular V we use function approximators.

Overview

Table of Contents

Temporal Difference Learning

This lecture: Model-free prediction and control. Estimate/ optimize the value function of an *unknown* MDP using Temporal Difference Learning.

- TD is model-free: no knowledge required about MDP dynamics
- TD methods learn from episodes of *experiences*
  experiences = sequences of states, actions, and rewards
- TD learns from *incomplete* episodes by bootstrapping
- Bootstrapping: update estimated based on other estimates without waiting for a final outcome (update a guess towards a guess)

Model-free Prediction

- Goal: learn the state-value function V^π for a given policy π

$$S_0, A_0, R_1, \ldots, S_T \sim \pi$$

- Recall: the *return* is the total discounted reward

$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-1} R_T$$

- Recall: the value function is the expected return

$$V^\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] = \mathbb{E}_\pi\left[R_{t+1} + \gamma V^\pi(S_{t+1}) | S_t = s\right]$$

- Idea: estimate V^π from experience by averaging the returns observed after visits to that state $\rightarrow$ Use empirical mean return instead of expected return
- Estimating V^π directly from G_t leads to *Monte Carlo methods* (not in the focus of this school). Estimating V^π from $R_{t+1} + \gamma V^\pi(S_{t+1})$ leads to *Temporal Difference methods*.

Incremental and Running Mean

- We can compute the mean of a sequence $x_1, x_2, \ldots$ incrementally:

$$
\begin{aligned}
\mu_k &= \frac{1}{k} \sum_{j=1}^{k} x_j \\
&= \frac{1}{k} \left(x_k + \sum_{j=1}^{k-1} x_j \right) \\
&= \frac{1}{k} \left(x_k + (k-1) \frac{1}{k-1} \sum_{j=1}^{k-1} x_j \right) \\
&= \frac{1}{k} (x_k + (k-1)\mu_{k-1}) \\
&= \mu_{k-1} + \frac{1}{k} (x_k - \mu_{k-1})
\end{aligned}
$$

Incremental and Running Mean

- Thus, we can update V incrementally by:

$$V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)}(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)),$$

  where $\frac{1}{N(s)}$ is the state-visitation counter
- Instead $\frac{1}{k}$, we can use step size α to calculate a running mean:

$$V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$

TD Prediction

Monte Carlo Update

Update value $V(S_t)$ towards the *actual* return G_t.

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$

α is a step-size parameter.

Simplest temporal-difference learning algorithm: $\mathrm{TD}(0)$

Update value $V(S_t)$ towards the *estimated* return $R_{t+1} + \gamma V(S_{t+1})$.

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

- $R_{t+1} + \gamma V(S_{t+1})$ is called the *TD target*
- $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ is called the *TD error*

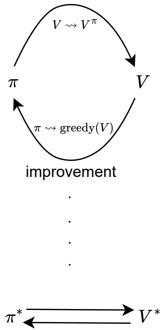
TD Prediction

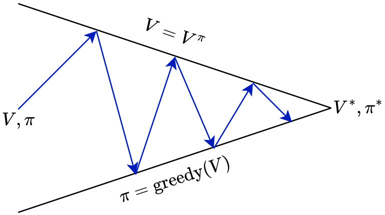
Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(\mathit{terminal}) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    $A \leftarrow$ action given by π for S
    Take action A, observe R, S'
    $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
    $S \leftarrow S'$
  until S is terminal

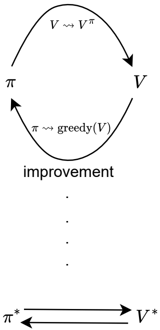
Generalized Policy Iteration

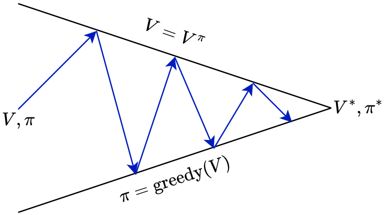




- Policy Evaluation: estimate V^π
- Policy Improvement: greedy

Generalized Policy Iteration with TD Evaluation





- Temporal Difference Policy Evaluation: $V \approx V^\pi$
- Policy Improvement: greedy?

TD Estimation of Action Values

- Greedy policy improvement over $V(s)$ requires a model of the MDP

$$\pi(s) = \arg\max_{a \in \mathcal{A}} \sum_{s'} P(s'|s,a)[r(s,a) + \gamma V(s')]$$

- Greedy policy improvement over $Q(s,a)$ is model-free

$$\pi(s) = \arg\max_{a \in \mathcal{A}} Q(s,a)$$

Generalized Policy Iteration with action-value function:

- Monte Carlo Policy Evaluation: $Q \approx Q^{\pi}$
- Policy Improvement: greedy?

ϵ-greedy Policy Improvement

- We have to ensure that each state-action pair is visited a sufficient (infinite) number of times
- Simple idea: ϵ-greedy
- All actions have non-zero probability
- With probability ϵ choose a random action, with probability $1 - \epsilon$ take the greedy action.

$$\pi(a \mid s) = \begin{cases} \frac{\epsilon}{|\mathcal{A}|} + 1 - \epsilon & \text{if } a = \arg\max_{a' \in \mathcal{A}} Q(s, a') \\ \frac{\epsilon}{|\mathcal{A}|} & \text{otherwise} \end{cases}$$

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*

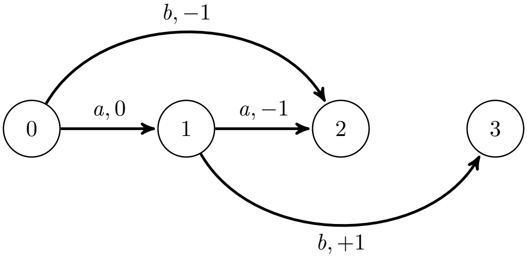
TD Control: Q-learning

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

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

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

Q-learning Example



$$
\begin{array}{lccccc}
\text{traj}_1: & 0 & \rightarrow & 1 & \rightarrow & 2 \\
\text{traj}_2: & 0 & \rightarrow & 1 & \rightarrow & 3 \\
\text{traj}_3: & 0 & \rightarrow & 1 & \rightarrow & 2
\end{array}
$$

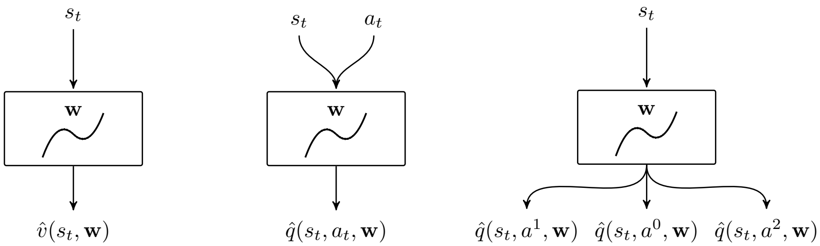
Table of Contents

Function Approximation in Reinforcement Learning

- Up to this point, we represented all elements of our RL systems by tables (value functions, models and policies)
- If the state and action spaces are very large or infinite, this is not a feasible solution
- We can apply function approximation to find a more compact representation of RL components and to generalize over states and actions
- Reinforcement Learning with function approximation comes with new issues that do not arise in Supervised Learning – such as non-stationarity, bootstrapping and delayed targets

Function Approximation in Reinforcement Learning

- Here: we estimate value-functions $V^{\pi}(\cdot)$ and $Q^{\pi}(\cdot,\cdot)$ by function approximators $\hat{v}(\cdot,\mathbf{w})$ and $\hat{q}(\cdot,\cdot,\mathbf{w})$, parameterized by weights $\mathbf{w}$



- But we can also represent models or policies

Function Approximation in Reinforcement Learning

We can use different types of function approximators:

- Linear combinations of features
- Neural networks
- Decision trees
- Gaussian processes
- Nearest neighbor methods
- ...

Here: We focus on differentiable FAs and update the weights via gradient descent.

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

$$\begin{aligned}\mathbf{w}_{t+1} &= \mathbf{w}_t - \frac{1}{2}\alpha\nabla\left[V^\pi(S_t) - \hat{v}(S_t,\mathbf{w}_t)\right]^2 \\ &= \mathbf{w}_t + \alpha\left[V^\pi(S_t) - \hat{v}(S_t,\mathbf{w}_t)\right]\nabla\hat{v}(S_t,\mathbf{w}_t)\end{aligned}$$

However, we do not have $V^\pi(S_t)$.

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*?

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!

Deep Q-Networks (DQN)

DQN provides a stable solution to deep RL:

- Use experience replay
- Sample minibatches (as opposed to full Batches)
- Freeze target Q-networks
- 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, S, S') from D
- Optimize MSE between Q-network and Q-learning targets, e.g.

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

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 $\mathbf{w}^-$

$$R + \gamma \max_{a'} Q(S', A', \mathbf{w}^-)$$

- Optimize MSE between Q-network and target network

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

- 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

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

This is your exercise

Initialize replay memory D to capacity N and Q-network weights $\mathbf{w}$ with $\mathbf{w}^- = \mathbf{w}$
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, S'_j, R_j) from D with batchsize B
    Compute target y_j for each sample $j = 1, \ldots, B$:

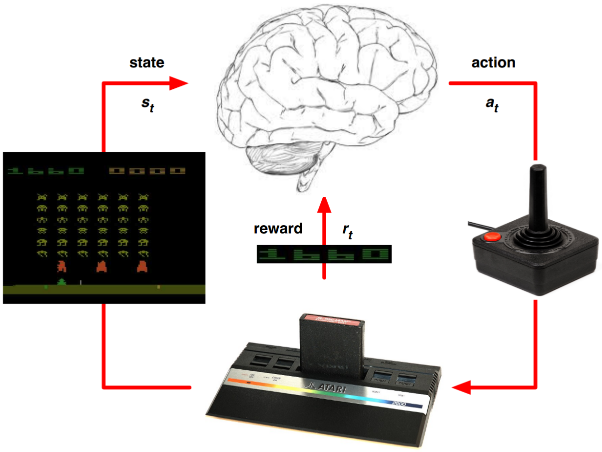
$$Y_j = \begin{cases} R_j & \text{if } S'_j \text{ is terminal} \\ R_j + \gamma \max_{a'} Q(S'_j, a', \mathbf{w}^-) & \text{else} \end{cases}$$

    Update the parameters of Q according to:

$$\nabla \mathbf{w} \widehat{L}(\mathbf{w}) = \frac{\alpha}{B} \sum_{j=1}^{B} \Big(Y_j - Q(S_j, A_j, \mathbf{w}) \Big) \nabla_{\mathbf{w}} Q(S_j, A_j, \mathbf{w})$$

    Update target network with hard or slow update
  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**

Summary by Learning Goals

- **Model-free learning:** Temporal Difference (TD) methods estimate value functions from experience, using bootstrapping to learn incrementally without a model of the environment.
- **Policy improvement and control:** TD control methods (e.g., Q-learning) enable learning optimal policies, often combined with exploration strategies like ϵ-greedy and off-policy updates.
- **Function approximation and deep RL:** Large or continuous state spaces require function approximators (linear, neural networks), with techniques like DQN using experience replay and target networks for stability.