Tabular Reinforcement Learning Methods Seminar

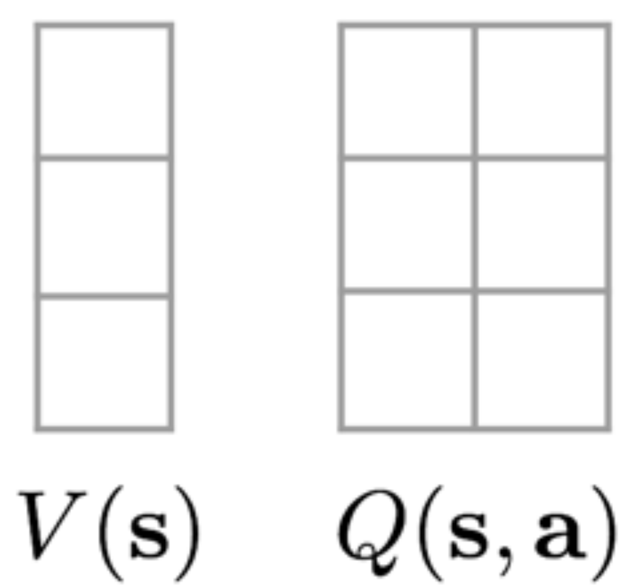
Oleksii Hrinchuk

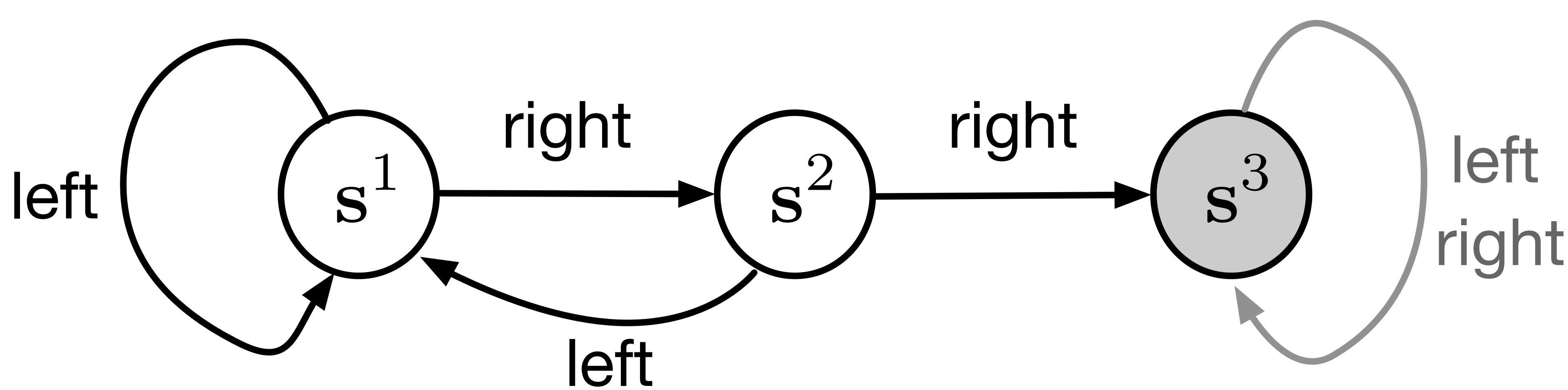
Tabular RL

- Discrete state $\mathcal{S}$ and action $\mathcal{A}$ spaces

- Value function (Q-function) is represented as vector (matrix) which stores corresponding values for all states (state-action pairs)

$$V \in \mathbb{R}^{|\mathcal{S}|}, \quad Q \in \mathbb{R}^{|\mathcal{S}| \times |\mathcal{A}|}$$





Model-based methods

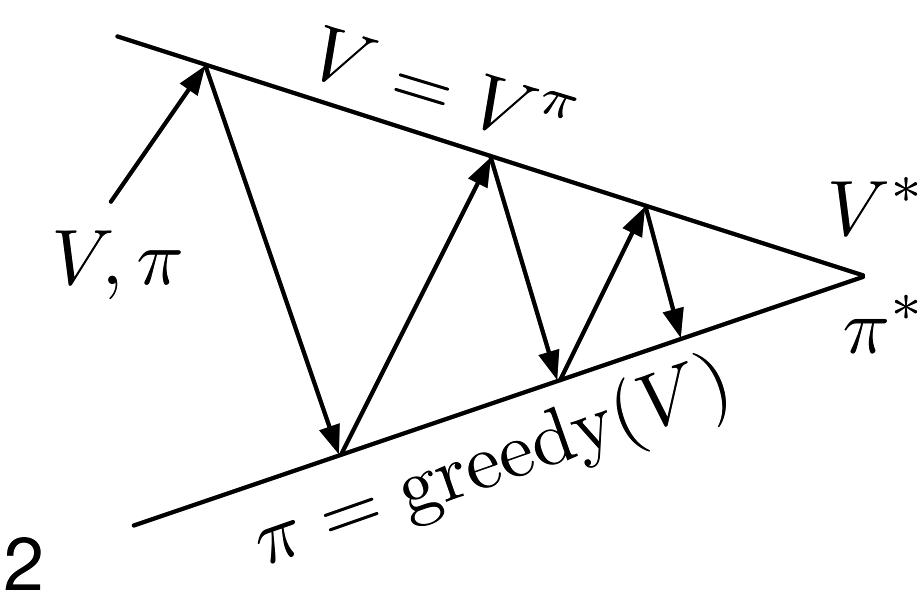
DP Policy Iteration

1. Set initial policy π (e.g. random policy)

2. (Iterative policy evaluation)
   — Initialize value function V_0^π (e.g. zeros)
   — Repeat
   for each state $\mathbf{s} \in \mathcal{S}$:
   $$V_{k+1}^\pi(\mathbf{s}) = \mathbb{E}_{\mathbf{a}}\left[r(\mathbf{s},\mathbf{a}) + \gamma \mathbb{E}_{\mathbf{s}'} V_k^\pi(\mathbf{s}')\right]$$
   $$\Delta = \max_{\mathbf{s}\in\mathcal{S}} \left| V_{k+1}^\pi(\mathbf{s}) - V_k^\pi(\mathbf{s}) \right|$$
   $$k \leftarrow k+1$$
   until $\Delta < \varepsilon$

3. (Greedy policy improvement)
   $$\pi \leftarrow \mathrm{greedy}(V_{k+1}^\pi)$$

4. If policy is good, return π, else go to step 2



Value Iteration

1. Initialize value function V_0 (e.g. zeros)

2. (Value iteration)
   — Repeat
   for each state $\mathrm{s} \in \mathcal{S}$:
   $$V_{k+1}(\mathbf{s}) = \max_{\mathbf{a} \in \mathcal{A}} \left[r(\mathbf{s}, \mathbf{a}) + \gamma \mathbb{E}_{\mathbf{s}'} V_k(\mathbf{s}') \right]$$
   $$\Delta = \max_{\mathbf{s} \in \mathcal{S}} |V_{k+1}(\mathbf{s}) - V_k(\mathbf{s})|$$
   $$k \leftarrow k + 1$$
   until $\Delta < \varepsilon$

3. Return greedy policy w.r.t. V_{k+1}

$$\pi(\mathbf{a}|\mathbf{s}) = \begin{cases} 1, & \mathbf{a} = \arg\max_{\mathbf{a} \in \mathcal{A}} \left[r(\mathbf{s}, \mathbf{a}) + \gamma \mathbb{E}_{\mathbf{s}'} V_{k+1}(\mathbf{s}') \right] \\ 0, & \text{otherwise} \end{cases}$$

Model-free methods

MC Policy Iteration

1. Set initial policy π (e.g. random policy), initialize Q-function Q (e.g. zeros)

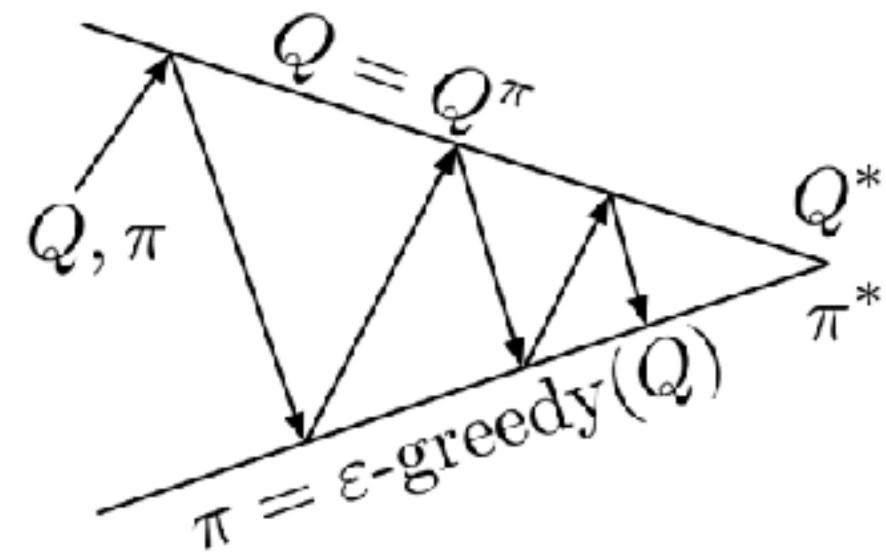
2. (Monte-Carlo policy evaluation)
   — Generate an episode $\tau_\pi = \{\mathbf{s}_0, \mathbf{a}_t, \mathbf{r}_0, \mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T\}$ using π
   — for each time step $t = 0, 1, \ldots, T-1$

   $$Z^\pi(\mathbf{s}_t, \mathbf{a}_t) = \mathbf{r}_t + \gamma \mathbf{r}_{t+1} + \ldots$$

   $$Q(\mathbf{s}_t, \mathbf{a}_t) \leftarrow Q(\mathbf{s}_t, \mathbf{a}_t) + \alpha[Z^\pi(\mathbf{s}_t, \mathbf{a}_t) - Q(\mathbf{s}_t, \mathbf{a}_t)]$$

3. (Epsilon-greedy policy improvement)

   $$\pi(\mathbf{a}|\mathbf{s}) \leftarrow \begin{cases} 1-\varepsilon, & \mathbf{a} = \arg\max_{\mathbf{a}\in\mathcal{A}} Q(\mathbf{s}, \mathbf{a}) \\ \varepsilon/(|\mathcal{A}|-1), & \text{otherwise} \end{cases}$$



4. If policy is good, return π, else go to step 2

   or greedy policy w.r.t. Q

TD Policy Iteration

1. Set initial policy π (e.g. random policy), initialize Q-function Q (e.g. zeros)

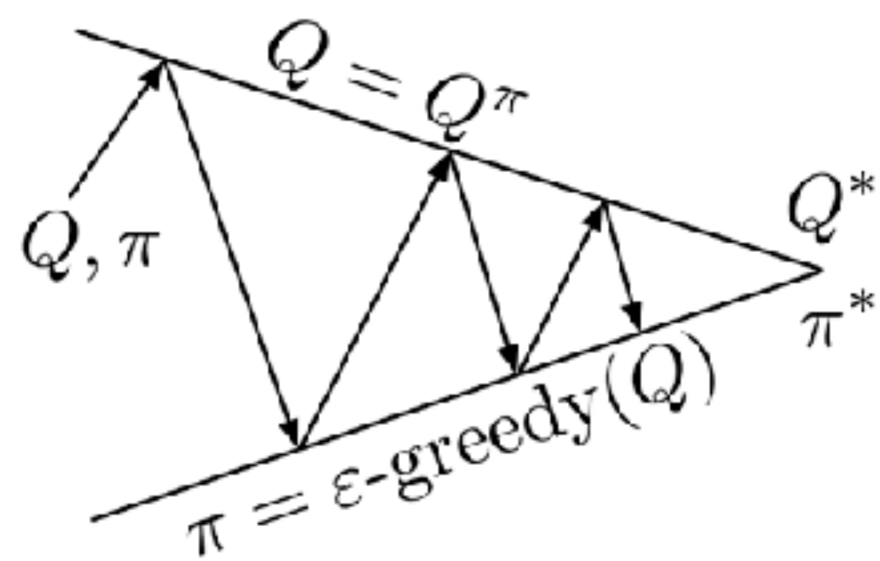
2. (SARSA policy evaluation step)
   — Make a step $\{\mathbf{s}_t, \mathbf{a}_t, \mathbf{r}_t, \mathbf{s}_{t+1}, \mathbf{a}_{t+1}\}$ in the environment using π
   — Update Q-values with TD-targets

$$y(\mathbf{s}_t, \mathbf{a}_t) = \mathbf{r}_t + \gamma Q(\mathbf{s}_{t+1}, \mathbf{a}_{t+1})$$

$$Q(\mathbf{s}_t, \mathbf{a}_t) \leftarrow Q(\mathbf{s}_t, \mathbf{a}_t) + \alpha[y(\mathbf{s}_t, \mathbf{a}_t) - Q(\mathbf{s}_t, \mathbf{a}_t)]$$

3. (Epsilon-greedy policy improvement step)

$$\pi(\mathbf{a}|\mathbf{s}) \leftarrow \begin{cases} 1-\varepsilon, & \mathbf{a} = \arg\max\limits_{\mathbf{a}\in\mathcal{A}} Q(\mathbf{s}, \mathbf{a}) \\ \varepsilon/(|\mathcal{A}|-1), & \text{otherwise} \end{cases}$$



4. If policy is good, return π, else go to step 2

or greedy policy w.r.t. Q

Online Q-learning

1. Set initial behavior policy π (e.g. random policy), initialize Q-function Q (e.g. zeros)

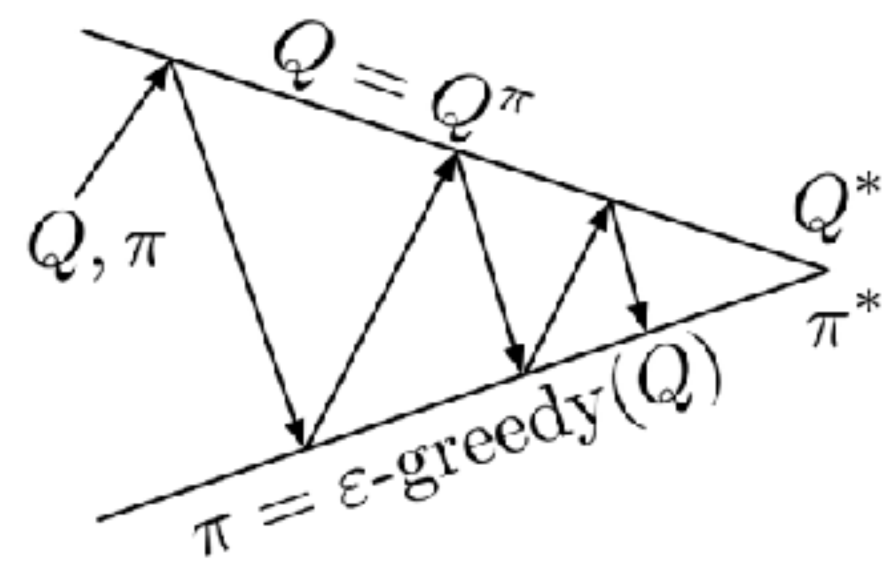
2. (Q-function update step)
   — Make a step $\{\mathbf{s}_t, \mathbf{a}_t, \mathbf{r}_t, \mathbf{s}_{t+1}\}$ in the environment using π
   — Update Q-values with TD-targets

$$y(\mathbf{s}_t, \mathbf{a}_t) = \mathbf{r}_t + \gamma \max_{\mathbf{a}' \in \mathcal{A}} Q(\mathbf{s}_{t+1}, \mathbf{a}')$$

$$Q(\mathbf{s}_t, \mathbf{a}_t) \leftarrow Q(\mathbf{s}_t, \mathbf{a}_t) + \alpha[y(\mathbf{s}_t, \mathbf{a}_t) - Q(\mathbf{s}_t, \mathbf{a}_t)]$$

3. (Behavior policy improvement)

$$\pi(\mathbf{a}|\mathbf{s}) \leftarrow \begin{cases} 1 - \varepsilon, & \mathbf{a} = \arg\max_{\mathbf{a} \in \mathcal{A}} Q(\mathbf{s}, \mathbf{a}) \\ \varepsilon / (|\mathcal{A}| - 1), & \text{otherwise} \end{cases}$$



4. If policy is good, return π, else go to step 2

or greedy policy w.r.t. Q

Problem set

1. Implement and test DP policy iteration
   Plot the dependence of average reward on number of iterations / value sweeps. **Note:** to estimate average reward of a given policy, run it ~100 times after each policy improvement step.

2. Implement and test Value Iteration algorithm
   Plot the dependence of value estimate error $\delta_k = \max_{\mathbf{s} \in \mathcal{S}} |V_k(\mathbf{s}) - V^*(\mathbf{s})|$ on number of value sweeps k . **Note:** ground truth value function V^* can be obtained from task 1.

3. Implement and test MC and TD policy iteration algorithms
   Plot the dependence of average reward on number of Q-function updates. **Note:** you can either use rewards obtained with epsilon-greedy policy during data collection, or run greedy policy for ~100 times after every let's say 1000 updates of Q-function.

Bonus problem

Pick favorite model-free RL method and apply to your environment

1. Plot the dependence of average reward on number of transitions made in the environment

2. Play with the size of your problem and the level of stochasticity in the environment and report the results of experiments in a form of a table or a plot.

3. Find parameters of the problem (size and stochasticity) when your off-policy methods fail (or take more than 3 minutes to find the solution). Add new feature to your environment which allows to run it in model-based setting. Solve the problem you did not manage to solve with your favorite model-based RL method.