# Reinforcement Learning - Tutorial

W.J.A. van Heeswijk

MSCA Digital Finance

Training week on
Reinforcement Learning in
Digital Finance

3.2.2025

# Markov Decision Processes

# Problem setting [1/2]

## Building blocks of Markov Decision Process

- **State** $s \in \mathcal{S}$: Current system information, needed to compute next state, rewards and actions
- **Action** $a \in \mathcal{A}$: Response to the state, provided by policy $\pi : s \to a$
- **Reward** $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$: Reward or penalty $r(s, a)$ after performing action $a$ in state $s$
- **Transition** $\mathcal{P} : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$: Function to guide time step, based on state, action and exogenous information
- **Discount factor** $\gamma \in [0, 1]$: Discount of future rewards, both for convergence and emphasis on present rewards

# Problem setting [2/2]

## Other problem properties

- **Policy** $\pi : s \rightarrow a$: Mapping from state to action, which we must learn
- **Exogenous variable** $W$: Stochastic information revealed over time (not in this environment)
- **Objective function** $J$:

$$J(\theta) = E_{\tau \sim \pi}[R(\tau)]$$

# Dynamic programming solution

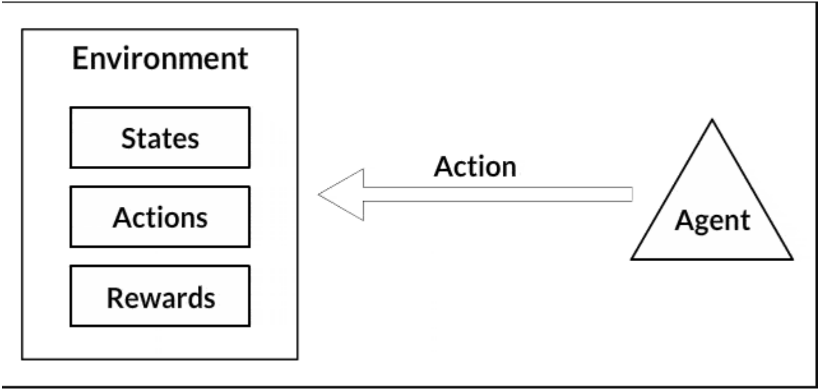
Solve system of Bellman equations:

$$V(s) = \max_a \left( R(s,a) + \gamma \sum_{s'} P(s,a,s')V(s') \right)$$

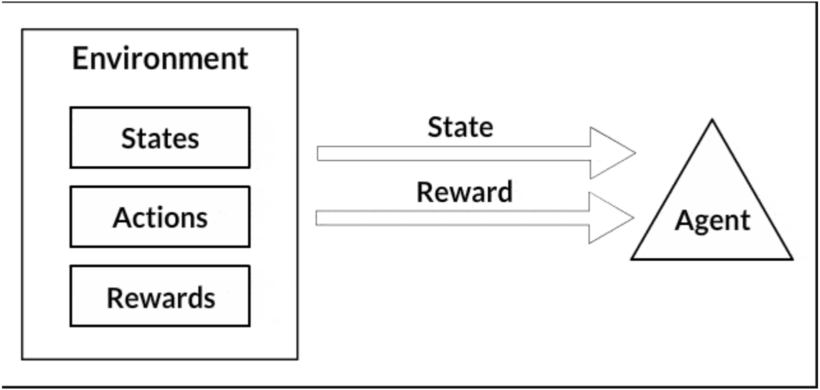
- Can be solved to optimality, but fails for large problems.
- In this tutorial, we try to *learn* the value functions.

# Q-learning

# Model-free learning [1/2]



# Model-free learning [2/2]



# Q-learning algorithm [1/2]

**Algorithm 1:** Q-learning algorithm

Fix parameters: learning rate $\alpha \in (0, 1]$, $\epsilon \in [0, 1]$ discount rate $\gamma \in [0, 1]$;
Initialize $Q(s, a) = 0, \forall s \in \mathcal{S}, a \in \mathcal{A}$;
**foreach** *episode* $n \in N$ **do**
  Initialize state $s$;
  **foreach** *epoch* $t \in \mathcal{T}$ **do**
    Sample $\hat{\epsilon} \in [0, 1]$;
    **if** $\hat{\epsilon} \leq \epsilon$ **then**
      Sample random action $a \in \mathcal{A}$;
    **end if**
    **else if** $\hat{\epsilon} > \epsilon$ **then**
      Select action $a = \arg\max_{a \in \mathcal{A}} r(s, a) + Q(s, a)$;
    **end if**
    $Q(s, a) \leftarrow Q(s, a) + \alpha[r(s, a) + \gamma \max_{a' \in \mathcal{A}} Q(s', a') - Q(s, a)]$;
    Update state $s \leftarrow s'$;
  **end foreach**
**end foreach**

# Q-learning algorithm [2/2]

Update equation:

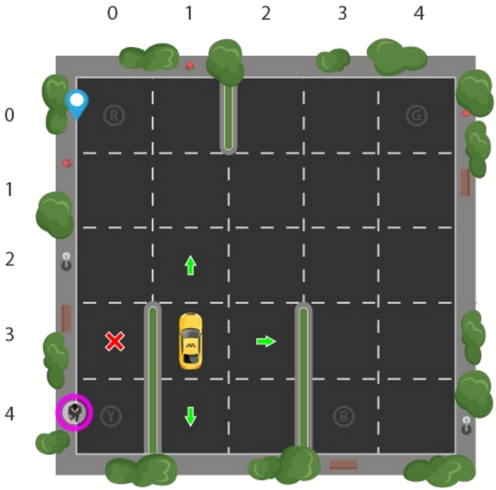
$$Q^{new}(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{current value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\bigg( \underbrace{\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}_{\text{new value (temporal difference target)}} - \underbrace{Q(s_t, a_t)}_{\text{current value}} \bigg)}^{\text{temporal difference}}$$

## What is important here?

- We select random action with probability $\epsilon$
- Off-policy: We use max function to find action next time step
  - Lower adverse impact of exploration
- Remember: just a simple weighted average

# Taxi Cab environment

# Taxi Cab Environment



# State space $\mathcal{S}$

**Exercise: setting up the state space**

- What information do we need to put in the state?
- How large is the state space?

# State space $\mathbb{S}$

## State components

- Location of taxi: 5 E 5
- Passenger location: 4 corners + 1 in taxi
- Passenger destination: 4 corners

State representation:

$$S = (\text{x\_pos\_taxi}, \text{y\_pos\_taxi}, \text{passenger\_loc}, \text{passenger\_dest})$$

State space $\mathbb{S}$ comprised of $5 \cdot 5 \cdot (4 + 1) \cdot 4 = 500$ states

# Action space $\mathbb{A}$

## Action

1. Move down
2. Move up
3. Move right
4. Move left
5. Pick up passenger
6. Drop off passenger

- Just one action at a time here, so small action space $\mathbb{A}$.
- Not all actions are feasible!
- We may filter infeasible actions with a mask (Boolean array)

# Reward function $R$

## Reward structure

- Successful drop-off: high positive reward
- Drop-off at wrong location: high negative reward
- Move without drop-off: small negative reward

Why the latter?

# Transition dynamics $\mathbb{P}$

## Direct consequence of actions

- Move to next square
- Passenger picked up
- Passenger dropped off

No stochastic elements in transition!
What uncertainties could we encounter in real-world taxi fleet management?

# Discount rate $\gamma$

## Setting a discount rate

- Set discount rate $\gamma \in [0, 1]$
- Finite horizon problem, so not strictly needed
- To what extent do present actions impact future rewards?

**Time to code!**

# Setting up the notebook

## Setup steps

- Access Notebook via **https://shorturl.at/kG38t**
- Install necessary libraries (uncomment by removing #)
- Press Ctrl+Enter to run cell
- Import libraries

# Sanity check

## Creating our environment

- Run cell with run_animation function (Environment initialization)
- You should see
  - State space: Discrete(500)
  - Action space: Discrete(6)
  - State: [0-499]
  - Action: [0-5]
  - Action mask: Binary vector, e.g., [1, 1, 1, 0, 0, 0, 0]
  - Reward: -1, -10 or +20
  - A frame of the environment

# Simulating with random agent [1/2]

- Agent with random actions
- Starting point for Q-learning, no clue about good actions

## Exercise: complete the code

- Stop loop when successfully completing episode
- Randomly sample an action (hint: use *env.action_space.sample()*)
- Run animation when complete

What do you observe?
What happens when car is standing still?
Who has largest number of epochs?

# Simulating with random agent [2/2]

**Exercise: complete the code**

- Add stop after max. 100 epochs
- Add mask to block infeasible actions (hint: see documentation at top of notebook!)

- Number of epochs likely goes down (but still undirected actions)
- Number of failed dropoffs should get to 0

# Training the agent

- Learn decision-making policy using Q-learning
- Combination of exploitation (taking best action according to Q-table) and exploration (taking random action)

**Exercise: complete the code**

- Set appropriate values for $\epsilon$, $\gamma$, $\alpha$ and $N$
- Design action selection mechanism (you can ignore masks)
- Implement weighted update rule

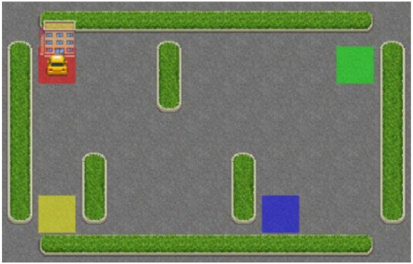
Does the policy converge?
What settings work well?

# Testing the policy

## Exercise: run the policy

- Run the learned policy
- Observe new animations

How does the agent make decisions?



# Wrapping up

# Results


- We learned a good policy for a toy problem
- Note the large number of observations (computational resources) required
- Every time we update the policy (i.e., the Q-table), we must create new observations again

# Possible extensions

- Managing fleet of taxis (send which one where?)
- Multi-agent learning, what do other taxis do?
- Massive increase in state- and action space
- Handling uncertainties (travel times, cancelling passengers, etc.)

DIGITAL

Funded by
the European Union

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or Horizon Europe: Marie Skłodowska-Curie Actions. Neither the European Union nor the granting authority can be held responsible for them.

*This project has received funding from the European Union's Horizon Europe research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 101119635*