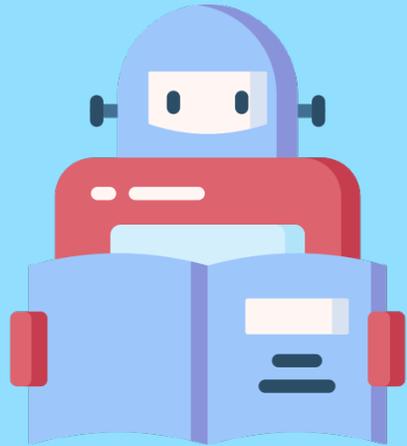


Mathematics for Machine Learning

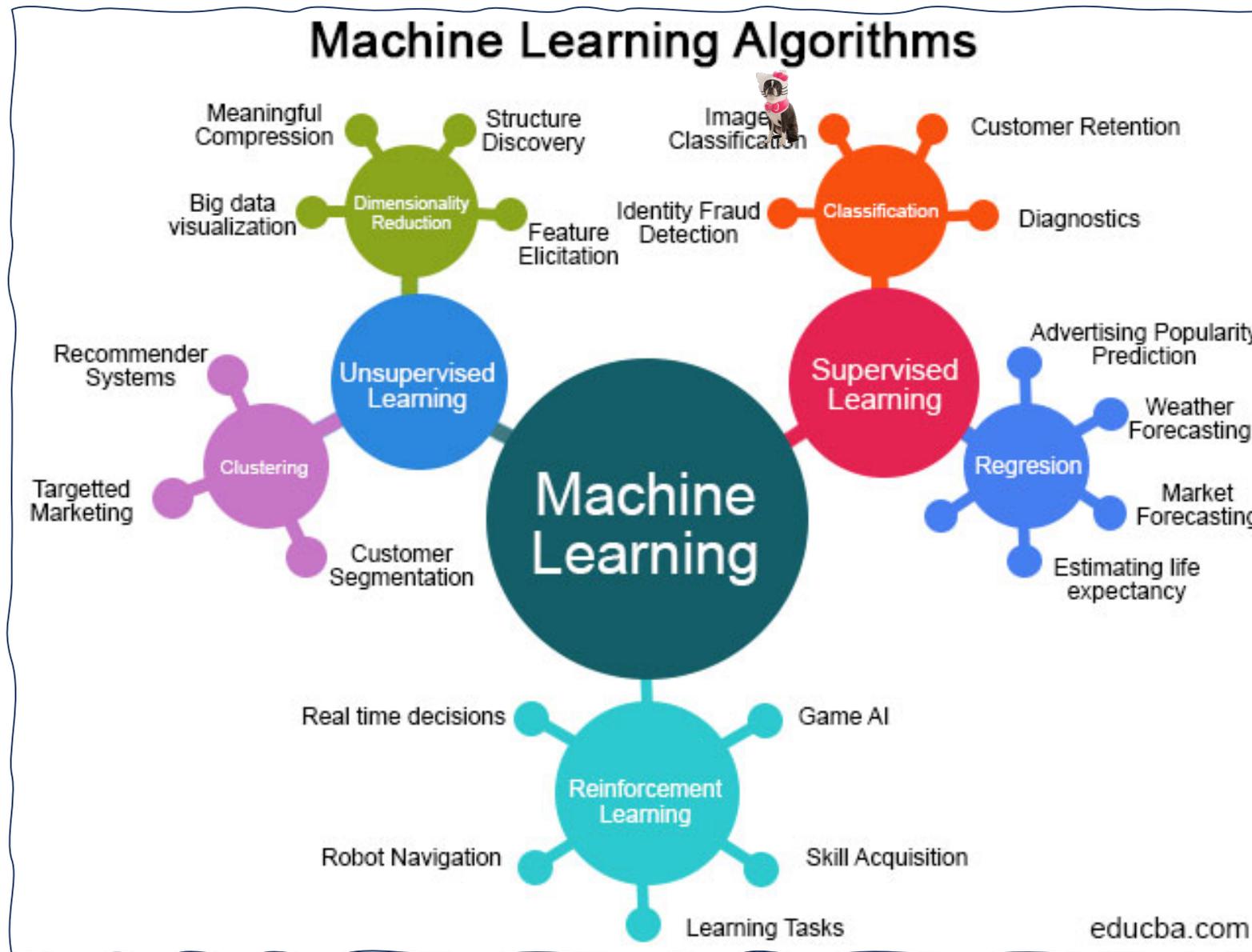


Special Mathematics Lecture
Nagoya University, Fall 2020

Lecture 9: Reinforcement Learning

https://www.henrikbachmann.com/mml_2020.html

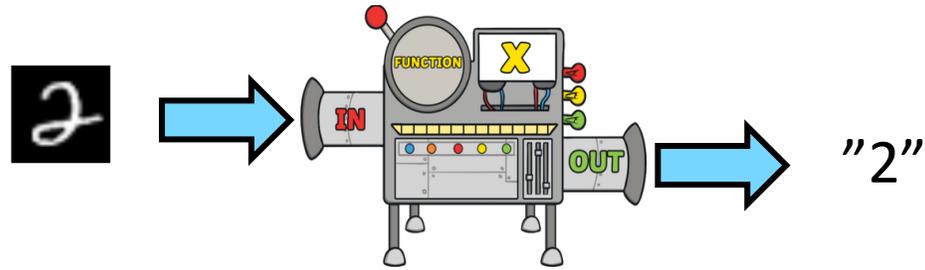
Recall: Machine learning overview



So far: Supervised learning

Classification (discrete output)

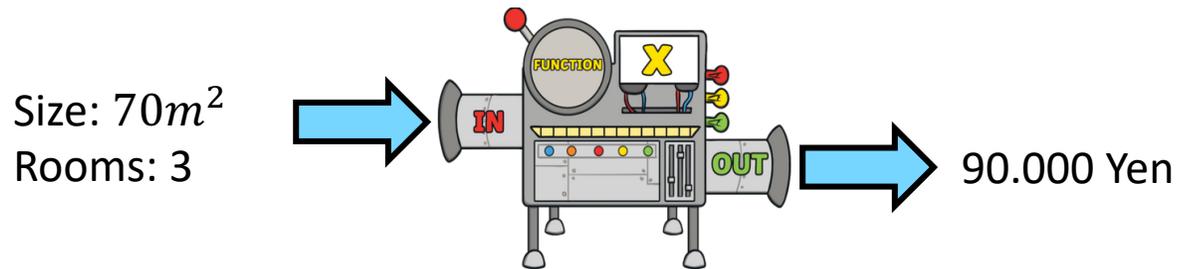
Output = category (e.g. "dog", "cat" or "1","2",...,"9")



Handwriting recognition

Regression (continuous output)

Output = real number (like "price", "weight",...)



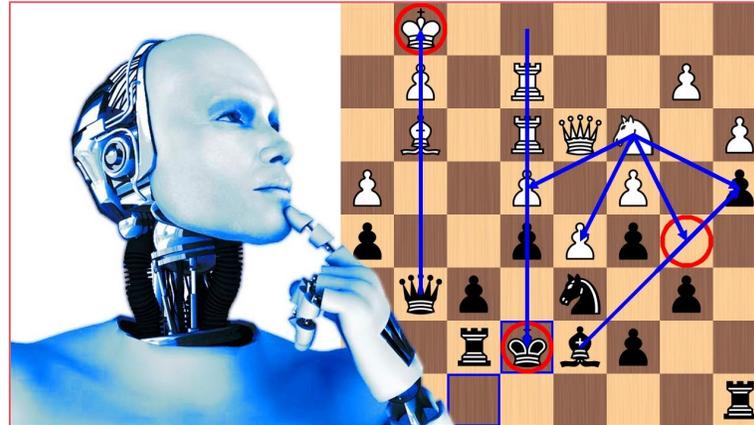
Apartment price prediction



Examples in this lecture so far

- Linear regression
- Logistic regression
- Naïve Bayes
- Support vector machines

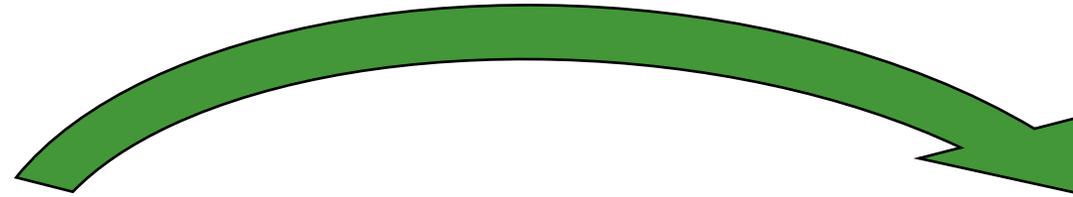
Supervised learning is not always possible



Reinforcement learning: Basic idea

Action

(move a chess piece, take a step, turn the wheel,...)



Reward

(win/lose the game, achieving a goal, falling over, crashing against a wall, ...)



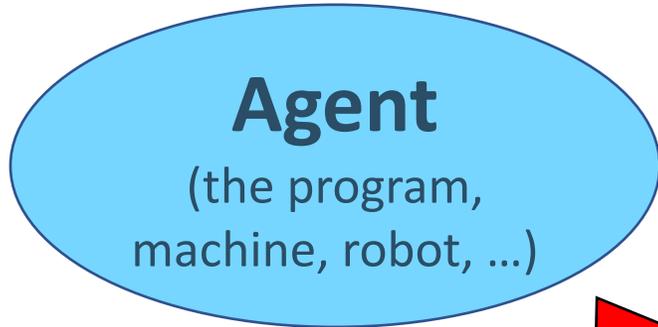
New state

(new position, new chessboard state, ...)



Agent

(the program, machine, robot, ...)



Environment



Markov decision process

A **Markov decision process (MDP)** is a tuple (S, A, T, R) , where

i) S is a set of states called the **state space**,

ii) A is a set of actions called the **action space**,

iii) T is a map

$$T : S \times A \times S \rightarrow [0, 1],$$

called the **transition probability function**,

iv) R is a map

$$R : S \times A \times S \rightarrow \mathbb{R},$$

called the **reward function**.

Markov decision process

A **Markov decision process (MDP)** is a tuple (S, A, T, R) , where

i) S is a set of states called the **state space**,

ii) A is a set of actions called the **action space**,

iii) T is a map

$$T : S \times A \times S \rightarrow [0, 1],$$

called the **transition probability function**,

iv) R is a map

$$R : S \times A \times S \rightarrow \mathbb{R},$$

called the **reward function**.

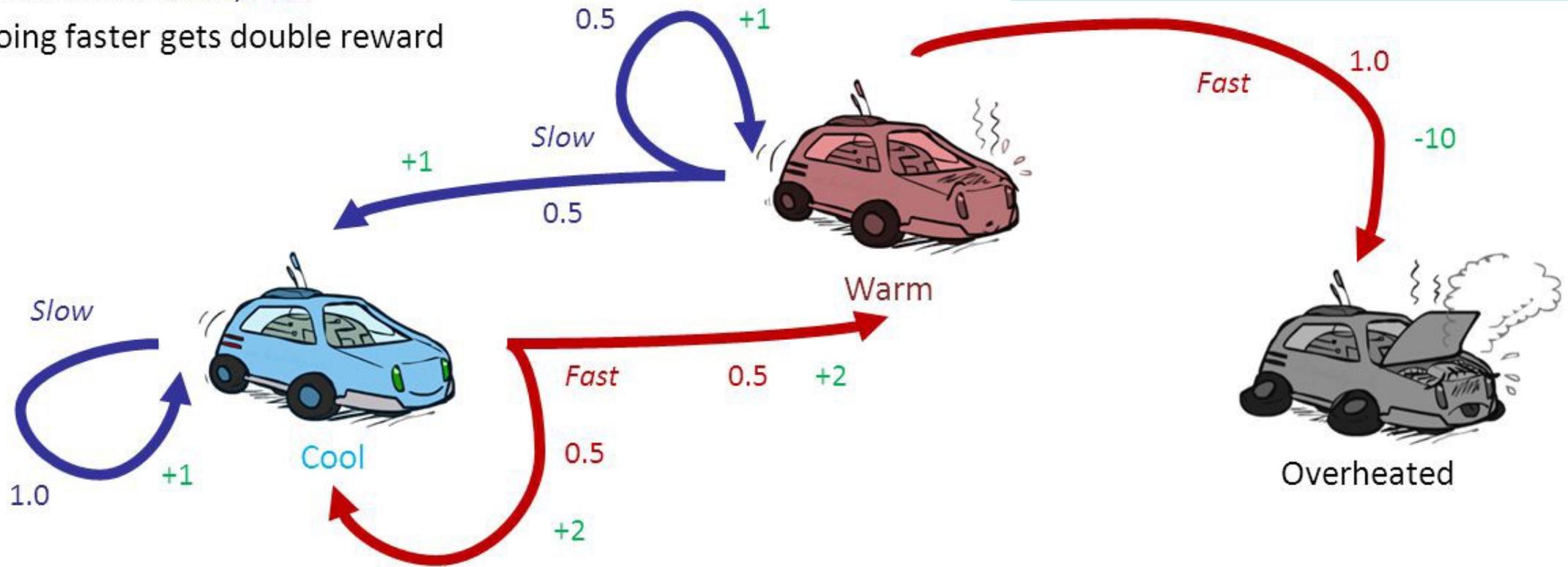
More precisely: For fixed $s \in S$ and $a \in A$ the transition probability function gives a probability distribution over S , i.e. for $s' \in S$

$$T(s, a, s') = P(s' \mid s, a).$$

$T(s, a, s')$: The probability that one reaches state s' from s when taking action a .

Markov decision process: Example

- A robot car wants to travel far, quickly
- Three states: **Cool**, **Warm**, **Overheated**
- Two actions: **Slow**, **Fast**
- Going faster gets double reward



A Markov decision process (MDP) is a tuple (S, A, T, R) , where

i) S is a set of states called the **state space**,

ii) A is a set of actions called the **action space**,

iii) T is a map

$$T : S \times A \times S \rightarrow [0, 1],$$

called the **transition probability function**,

iv) R is a map

$$R : S \times A \times S \rightarrow \mathbb{R},$$

called the **reward function**.

Markov decision process: Dynamics

- i) Start at some state $s_0 \in S$.
- ii) Choose an action $a_0 \in A$.
- iii) Obtain a new state $s_1 \in S$ (with probability $T(s_0, a_0, s_1)$) and a reward $R(s_0, a_0, s_1)$.
- iv) Repeat until one reaches a terminal state or a fixed number of steps T .

Goal: Try to choose the actions a_0, a_1, a_2, \dots at each step such that $\sum_{j \geq 0} R(s_j, a_j, s_{j+1})$ gets big.

Markov decision process: Dynamics & Discount

- i) Start at some state $s_0 \in S$.
- ii) Choose an action $a_0 \in A$.
- iii) Obtain a new state $s_1 \in S$ (with probability $T(s_0, a_0, s_1)$) and a reward $R(s_0, a_0, s_1)$.
- iv) Repeat until one reaches a terminal state or a fixed number of steps T .

Goal: Try to choose the actions a_0, a_1, a_2, \dots at each step such that $\sum_{j \geq 0} R(s_j, a_j, s_{j+1})$ gets big.

Problem: This could be an infinite sum and therefore one introduces a **discount factor** $\gamma \in [0, 1]$ and then considers the discounted total reward:

$$\sum_{j \geq 0} \gamma^j R(s_j, a_j, s_{j+1}).$$

Another interpretation: Immediate rewards count more than delayed rewards.

A strategy: Policy

A **policy** is a function $\pi : S \rightarrow A$.

Goal: Find the optimal policy which increases the reward.

A strategy: Policy

A **policy** is a function $\pi : S \rightarrow A$.

Goal: Find the optimal policy which increases the reward.

The **value of a policy** π at state $s \in S$ is defined by

$$V_{\pi}(s) = E \left[\sum_{j \geq 0} \gamma^j R(s_j, \pi(a_j), s_{j+1}) \mid s_0 = s \right]$$

Interpretation: The expected (discounted) total reward when starting in state $s_0 = s$ by using the policy π to choose the action $a_t = \pi(s_t)$ in state s_t .

Goal rephrased: Find the optimal policy which has maximal value at each state.

Optimal policy

A policy π^* is called **optimal** if it has maximal value for all states $s \in \mathcal{S}$:

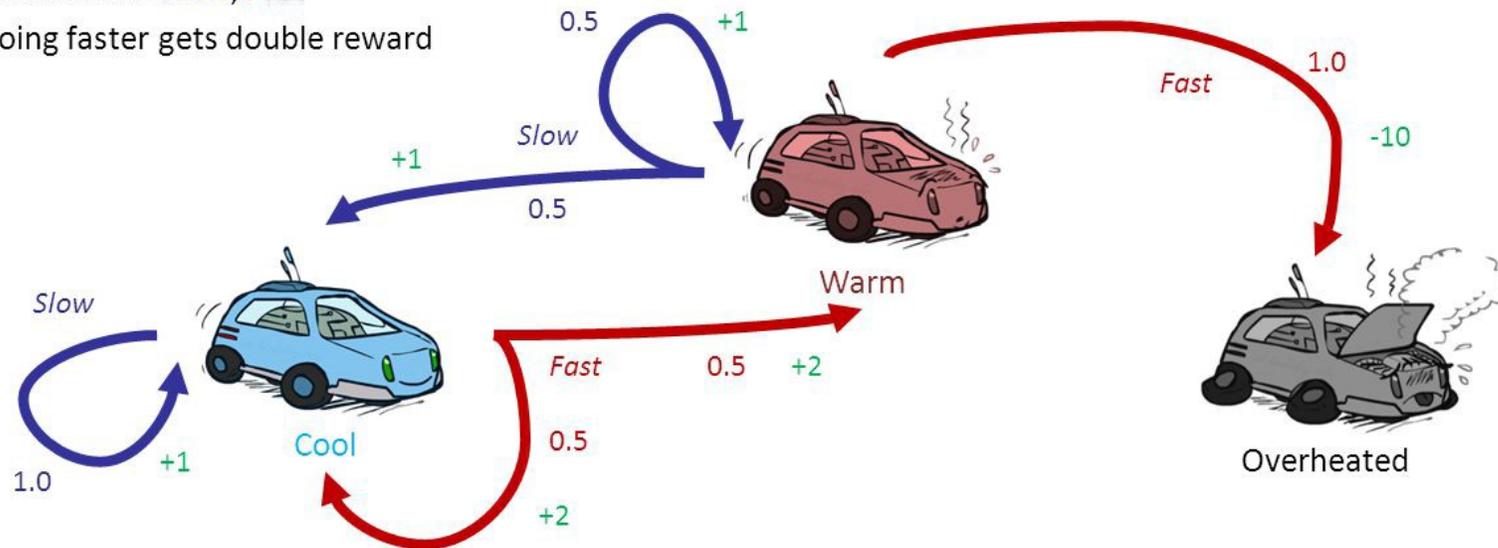
$$V_{\pi^*}(s) = \max_{\pi} V_{\pi}(s).$$

Optimal policy: Car example

A policy π^* is called **optimal** if it has maximal value for all states $s \in S$:

$$V_{\pi^*}(s) = \max_{\pi} V_{\pi}(s).$$

- A robot car wants to travel far, quickly
- Three states: **Cool**, **Warm**, Overheated
- Two actions: **Slow**, **Fast**
- Going faster gets double reward



Optimal policy & State-action value function

A policy π^* is called **optimal** if it has maximal value for all states $s \in S$:

$$V_{\pi^*}(s) = \max_{\pi} V_{\pi}(s).$$

The **state-action value function** Q^* is defined for all $(s, a) \in S \times A$ as the expected total reward for taking action $a \in A$ at state $s \in S$ following the optimal policy π^* :

$$Q^*(s, a) = \sum_{s' \in S} T(s, a, s')R(s, a, s') + \gamma \sum_{s' \in S} T(s, a, s')V_{\pi^*}(s')$$

Interpretation: This gives the best possible reward after choosing action a when in state s .

Goal: Find the values of the state-action value function

Q = quality

Optimal policy & State-action value function

A policy π^* is called **optimal** if it has maximal value for all states $s \in S$:

$$V_{\pi^*}(s) = \max_{\pi} V_{\pi}(s).$$

The **state-action value function** Q^* is defined for all $(s, a) \in S \times A$ as the expected total reward for taking action $a \in A$ at state $s \in S$ following the optimal policy π^* :

$$Q^*(s, a) = \sum_{s' \in S} T(s, a, s')R(s, a, s') + \gamma \sum_{s' \in S} T(s, a, s')V_{\pi^*}(s')$$

Interpretation: This gives the best possible reward after choosing action a when in state s .

Goal: Find the values of the state-action value function

Q = quality

Having the state-action value function Q^* we can derive the optimal policy by

$$\pi^*(s) = \operatorname{argmax}_{a \in A} Q^*(s, a).$$

Q-learning

The Q-learning algorithm tries to “learn” the state-action value function Q .

Q-learning algorithm: Find for all $s \in S$ and $a \in A$ a function $Q(s, a)$, which gives a good approximation for $Q^*(a, s)$.

1. Start with random values for $Q(s, a)$. (e.g. all zero)
2. Choose a starting state $s_0 \in S$.
3. Look up the current best action in that state, i.e. $a_0 = \operatorname{argmax}_{a \in A} Q(s_0, a)$.
4. Apply this action and get a new state s_1 and reward $r_0 = R(s_0, a_0, s_1)$.
5. Update the value $Q(s_0, a_0)$ as follows (**Bellman equation**)

$$Q(s_0, a_0) = (1 - \alpha)Q(s_0, a_0) + \alpha \left(r_0 + \gamma \max_{a \in S} Q(s_1, a) \right) .$$

Here $\alpha \in [0, 1]$ is the **learning rate**.

6. If s_1 is not a terminal state repeat with step 3.

Q-learning + Epsilon-Greedy

Q-learning algorithm: Find for all $s \in S$ and $a \in A$ a function $Q(s, a)$, which gives a good approximation for $Q^*(a, s)$.

1. Start with random values for $Q(s, a)$. (e.g. all zero)
2. Choose a starting state $s_0 \in S$.
3. Look up the current best action in that state, i.e. $a_0 = \operatorname{argmax}_{a \in A} Q(s_0, a)$ **or choose a random action $a_0 \in A$ with probability $\epsilon \in [0, 1]$ (Epsilon-Greedy Algorithm).**
4. Apply this action and get a new state s_1 and reward $r_0 = R(s_0, a_0, s_1)$.
5. Update the value $Q(s_0, a_0)$ as follows (**Bellman equation**)

$$Q(s_0, a_0) = (1 - \alpha)Q(s_0, a_0) + \alpha \left(r_0 + \gamma \max_{a \in S} Q(s_1, a) \right) .$$

Here $\alpha \in [0, 1]$ is the **learning rate**.

6. If s_1 is not a terminal state repeat with step 3.

Comбини & Lecture Example

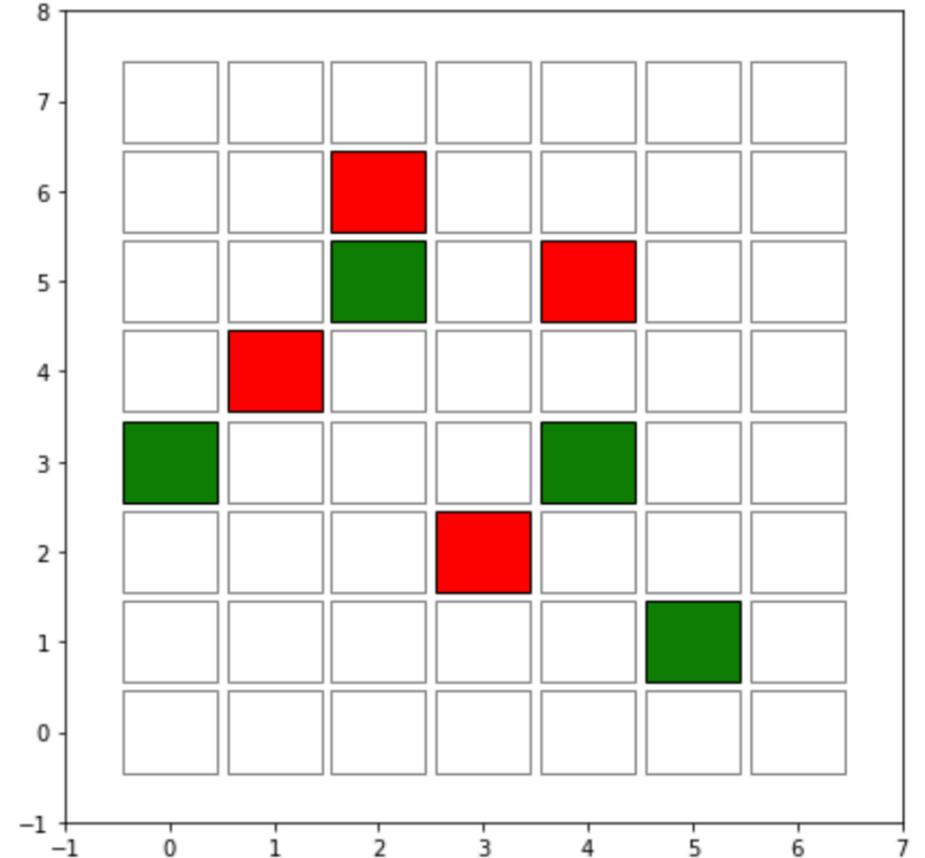
Goal: Find the nearest **Comбини** to get food. Avoid **lecture halls** so the professor does not know that you are actually on campus!

States: $7 * 8 = 56$ positions
(terminal states:  )

Actions:    

- Reaching Comбини: +1
- Reaching Lecture hall: -1
- Making a step to an empty position: -0.1

Optimal policy: Gives the direction we should go at each state to get to the nearest Comбини

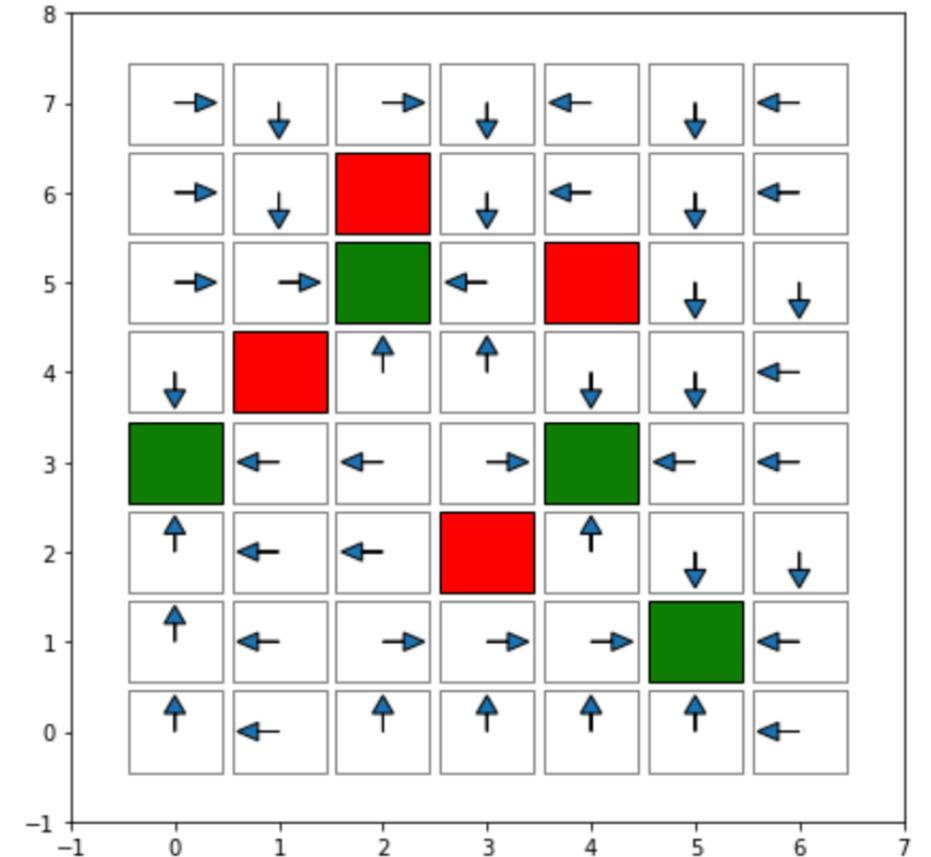


 **Lecture hall**
 **Comбини**

Comбини & Lecture Example: Possible optimal policy:

Goal: Find the nearest **Comбини** to get food. Avoid **lecture halls** so the professor does not know that you are actually on campus!

Optimal policy: Gives the direction we should go at each state to get to the nearest Comбини



 **Lecture hall**

 **Comбини**