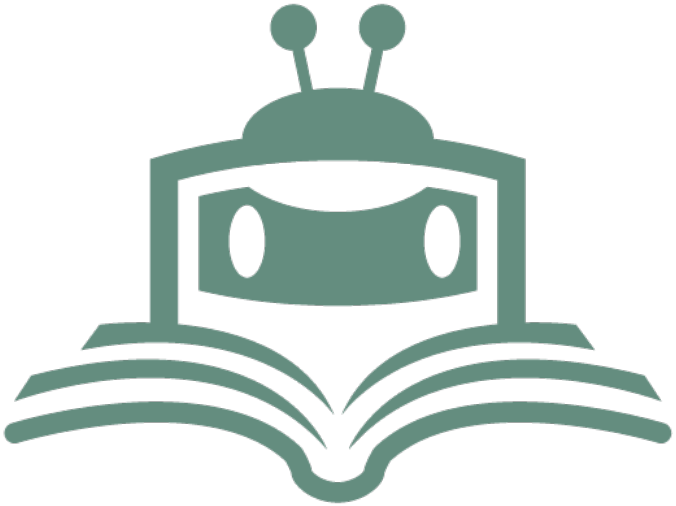


MATHEMATICS FOR MACHINE LEARNING

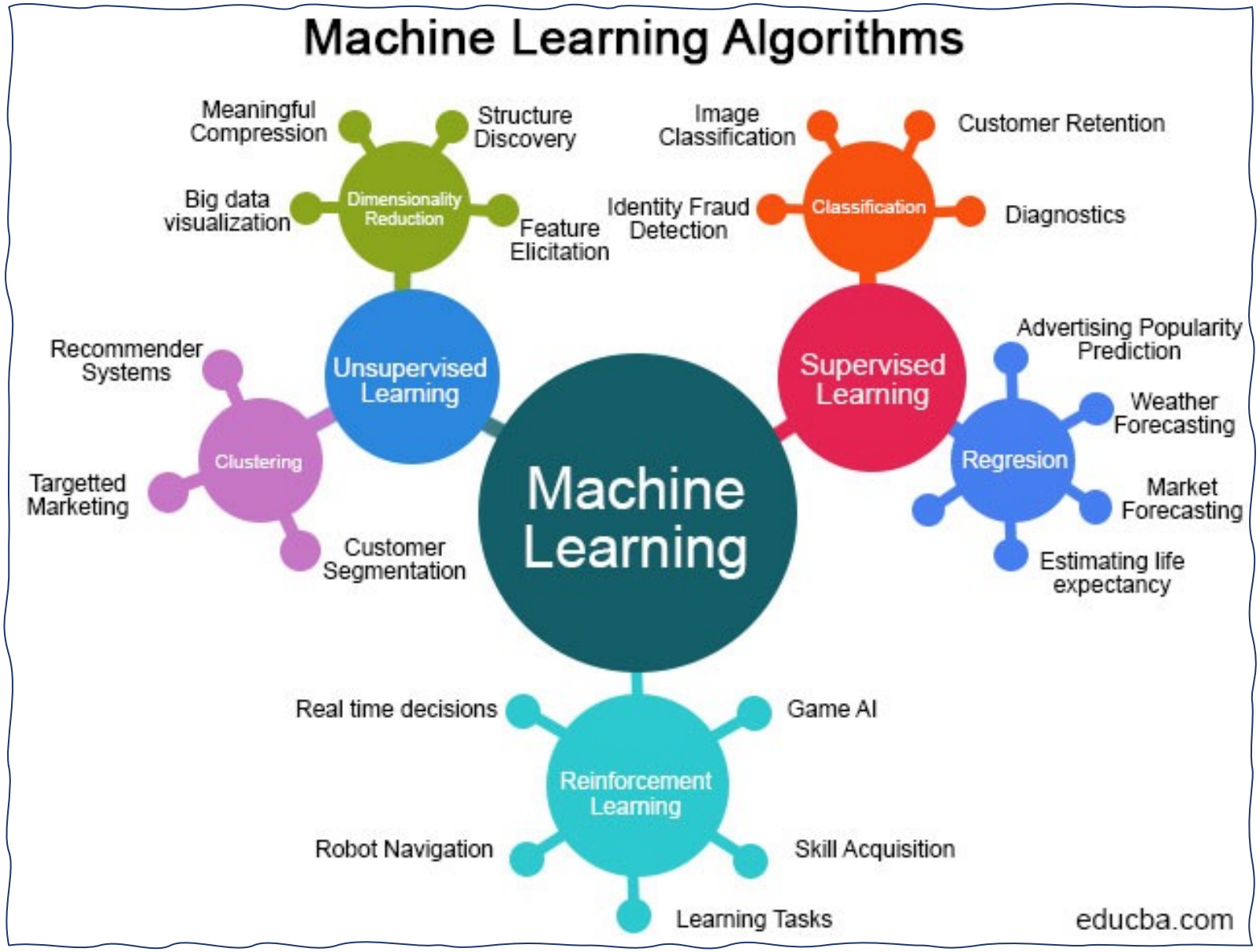
Nagoya University, Fall 2022

Lecture 9

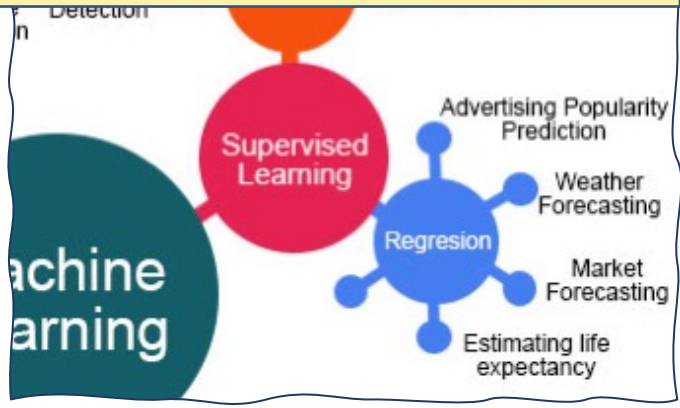
Reinforcement Learning: Q-learning

https://www.henrikbachmann.com/mml2022.html

Overview

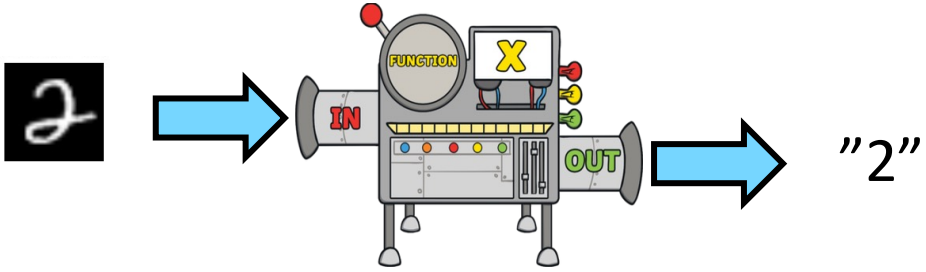

Machine Learning Algorithms
Machine Learning
Unsupervised Learning
Supervised Learning
Reinforcement Learning
Dimensionality Reduction
Classification
Clustering
Regresion
Meaningful Compression
Structure Discovery
Big data visualization
Feature Elicitation
Image Classification
Customer Retention
Identity Fraud Detection
Diagnostics
Recommender Systems
Targetted Marketing
Customer Segmentation
Advertising Popularity Prediction
Weather Forecasting
Market Forecasting
Estimating life expectancy
Real time decisions
Game AI
Robot Navigation
Skill Acquisition
Learning Tasks
educba.com

Supervised learning



Classification (discrete output)

Output = category (e.g. “dog”, “cat” or “1”,”2”,…,”9”)



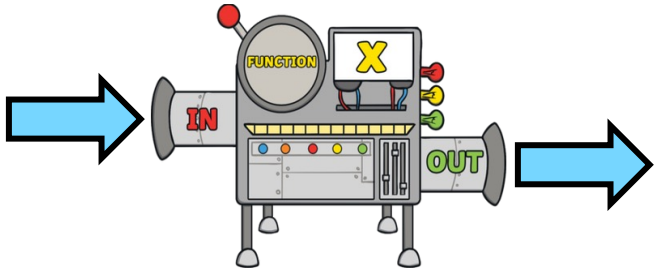
”2”

Handwriting recognition

Regression (continuous output)

Output = real number (like “price”, “weight”,…)

Size: $70m^2$
Rooms: 3



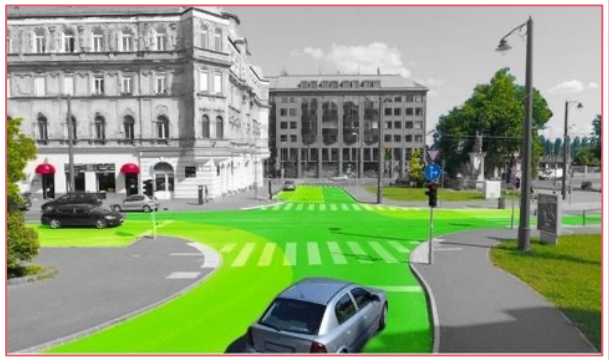
90.000 Yen

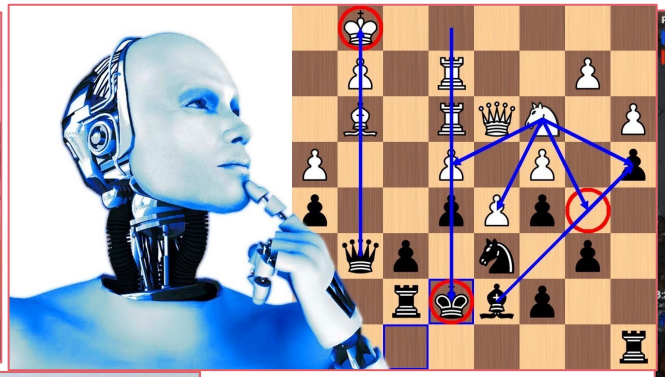
Apartment price prediction

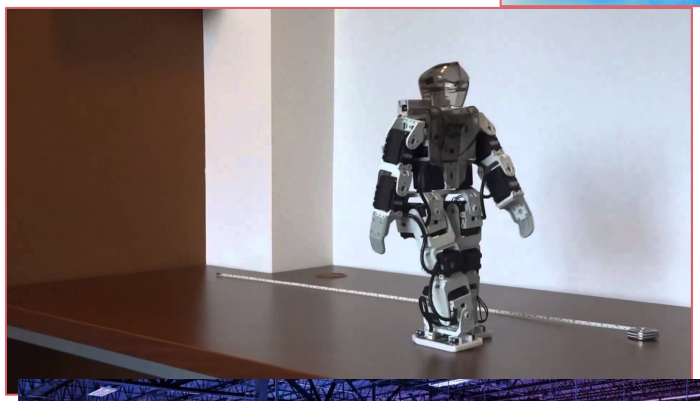
Examples in this lecture so far

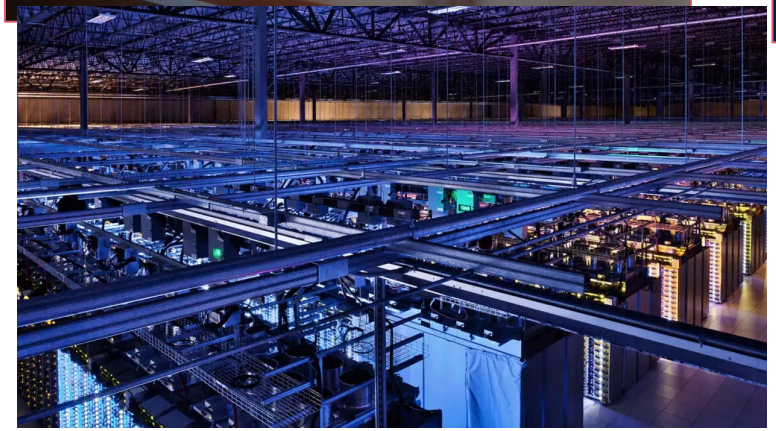
- Linear regression (Tebasaki)
- Logistic regression (Passing exam)
- Naïve Bayes (Spam filter)

Supervised learning is not always possible



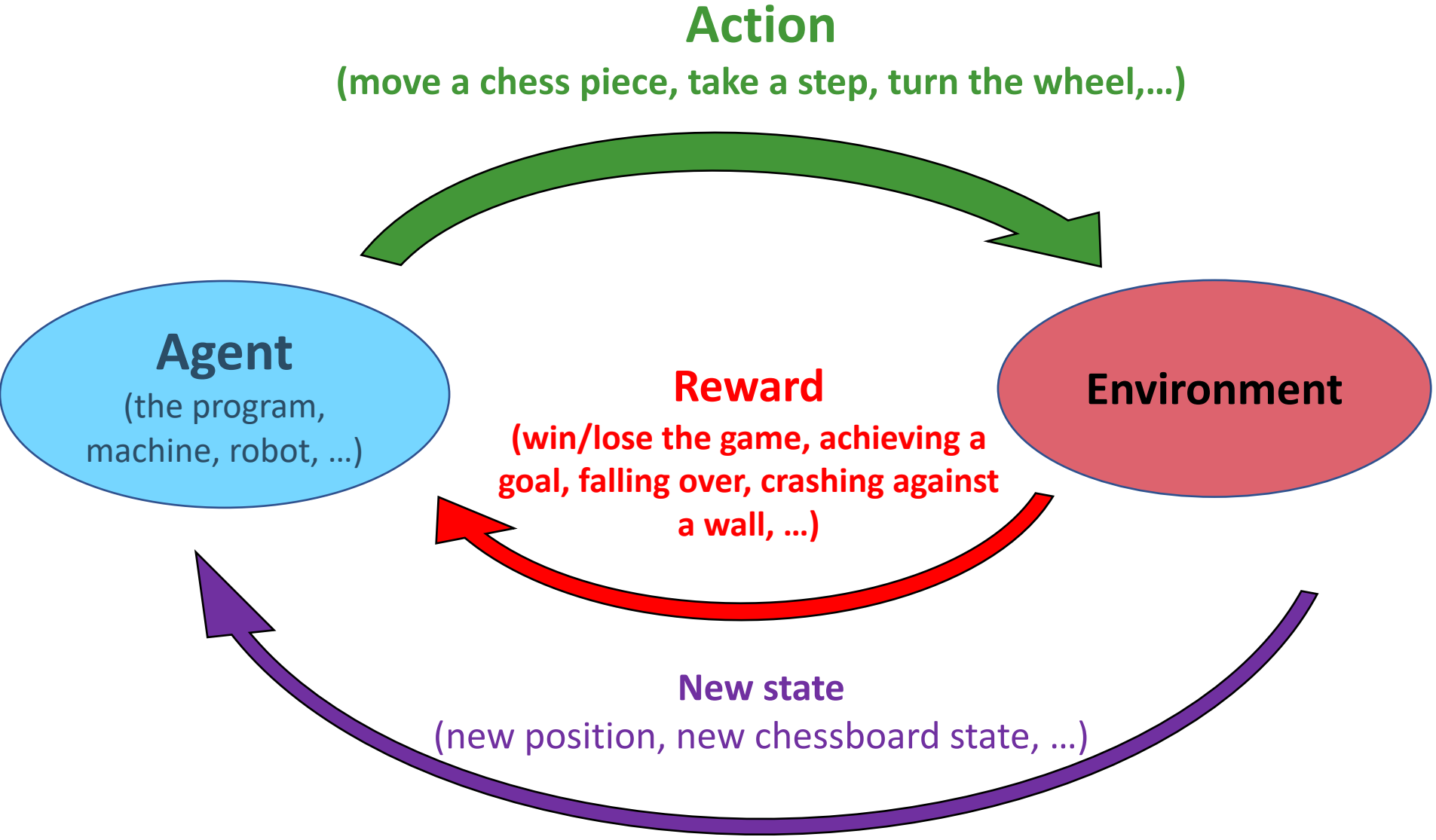






- Games: Chess, Go, RTS,…
- Movement of robots
- Selfdriving cars
- Google data center cooling
- ….

Reinforcement learning: Basic idea



Reinforcement learning: Walking example

https://www.youtube.com/watch?v=gn4nRCC9TwQ

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

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

called the **transition probability function**,

iv) R is a map

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

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

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

called the **transition probability function**,

iv) R is a map

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

called the **reward function**.

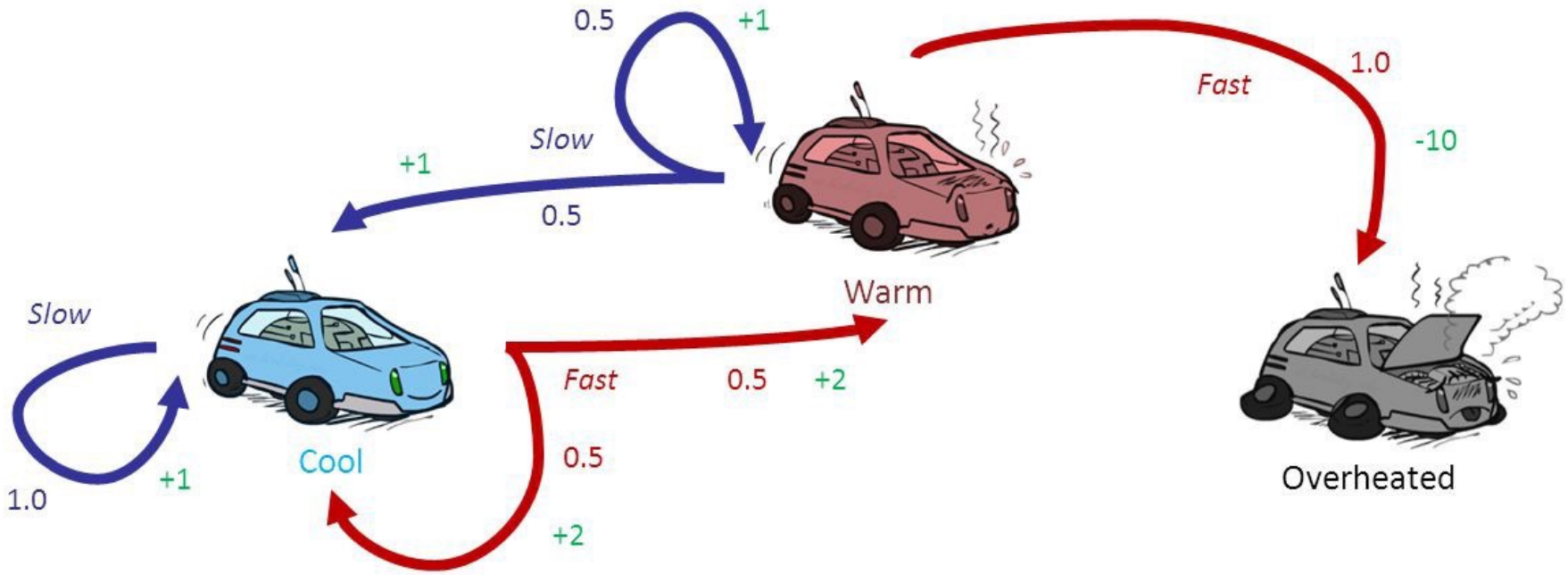
Interpretation:

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

$R(s, a, s')$: The reward that one gets by going from state s to s' by doing 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



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 N.

($N = \infty$ possible)

Goal: Choose the actions $a_0, a_1, a_2, \ldots$ at each state such that

$$\sum_{j=0}^{N} R(s_j, a_j, s_{j+1})$$

gets big.

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 N.

($N = \infty$ possible)

Goal: Choose the actions $a_0, a_1, a_2, \ldots$ at each state such that

$$\sum_{j=0}^{N} R(s_j, a_j, s_{j+1})$$

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:

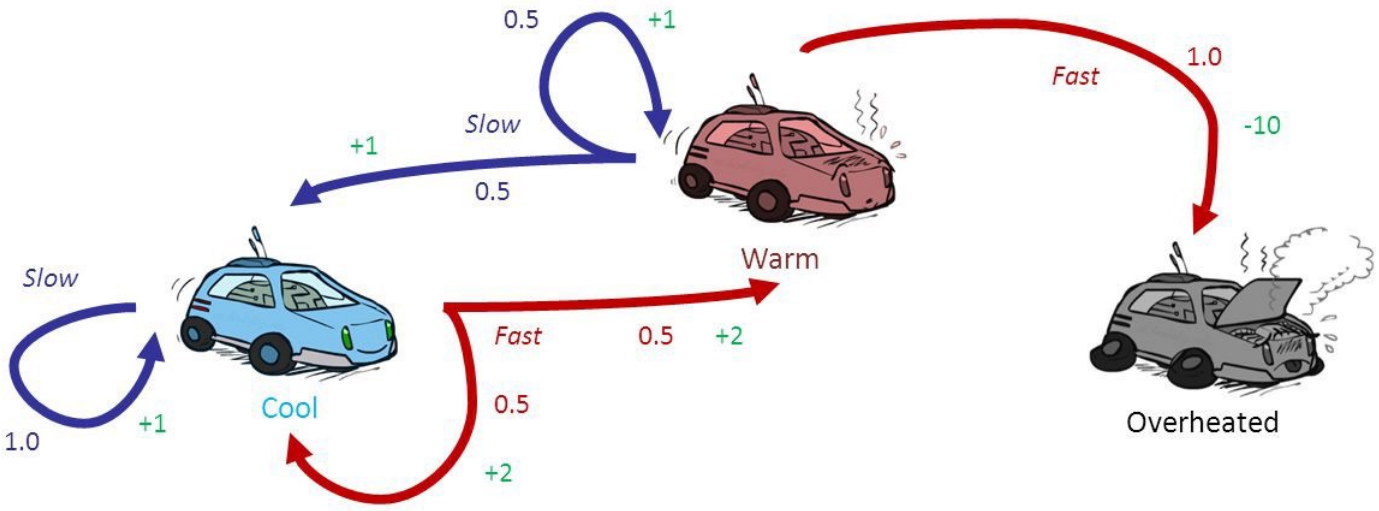
$$\sum_{j \geq 0} \gamma^j R(s_j, a_j, s_{j+1})\,.$$

Another interpretation: Immediate rewards count more than delayed rewards.

Total reward: Car example

For given sequences of states $(s_0, s_1, \dots)$ and actions $(a_0, a_1, \dots)$ the **discounted total reward** (with **discount** $\gamma \in [0, 1]$) is given by

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



Markov decision process: Policy

$$\text{A } \mathbf{policy} \text{ is a function } \pi : S \to A.$$

Interpretation: A policy suggests the action you should take when being in a certain state.

Goal: Find the optimal policy which increases the (discounted) total reward.

Markov decision process: Value of a policy

$$\text{A } \textbf{policy} \text{ is a function } \pi : S \to A.$$

Goal: Find the optimal policy which increases the (discounted) total reward.

$$\textit{The } \textbf{value of a policy } \pi \textit{ at state } s \in S \textit{ is defined by}$$

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

Markov decision process: Value of a policy

$$\text{A } \textbf{policy} \text{ is a function } \pi : S \to A.$$

Goal: Find the optimal policy which increases the (discounted) total reward.

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

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

Interpretation of $V_\pi(s)$**:** 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 a policy which has maximal value at each state.

Optimal policy

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

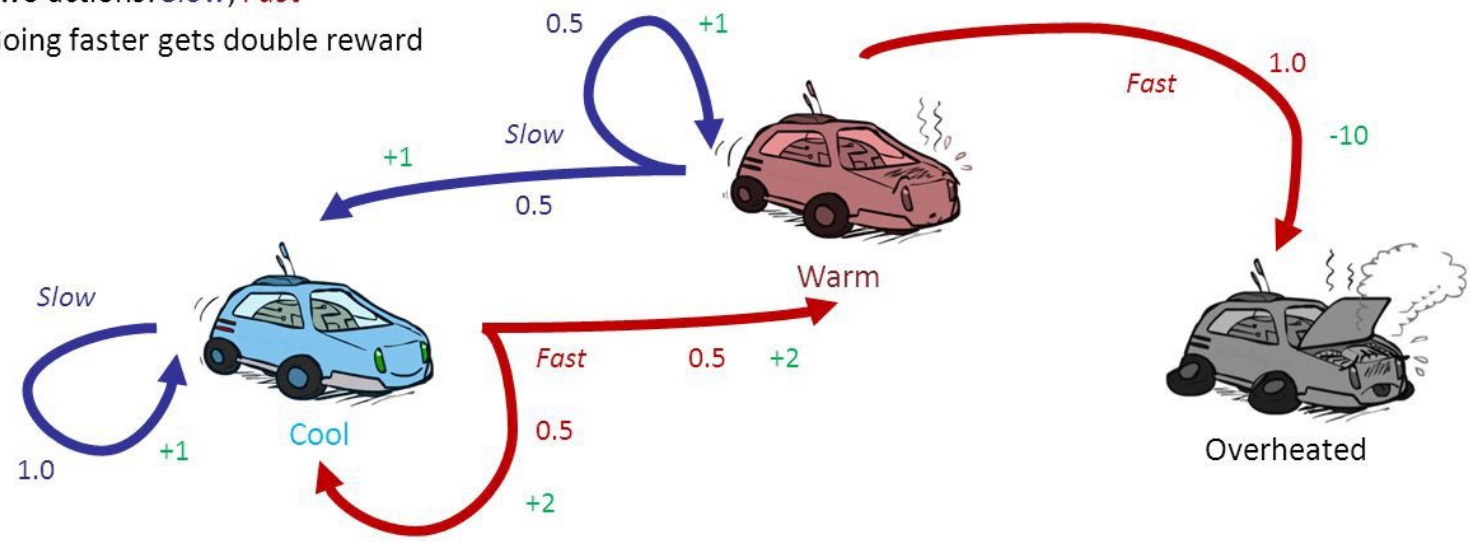
$$V_{\pi^*}(s) = \max_{\pi} V_{\pi}(s)\,.$$

Optimal policy

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

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

- Three states: Cool, Warm, Overheated
- Two actions: *Slow*, *Fast*
- Going faster gets double reward



Optimal policy & State-action value functio

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.

Optimal policy & State-action value functio

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

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

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

Optimal policy & State-action value functior

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

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

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

Goal rephrased: Find a good approximation of the state-action value function. “Learn” the function Q.

Q-learning

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. $a_0 = \text{argmax}_{a \in A} Q(s_0, a)$.
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 $Q^*(a, s)$.

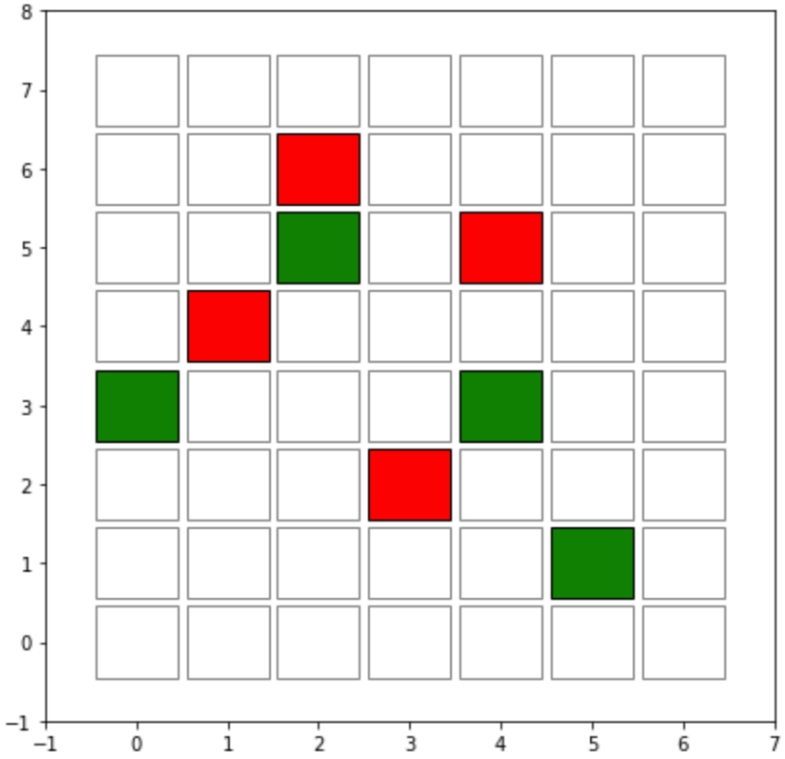
1. Start with random values for $Q(s, a)$. (e.g. all zero)
2. Choose a starting state $s_0 \in S$.
3. Look up the current best action in that state, i.e. $a_0 = \text{argmax}_{a \in A} Q(s_0, a)$ or choose a random action $a_0 \in A$ with probability $\epsilon \in [0, 1]$ (Epsilon-Greedy Algorithm).
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.

Combini & Lecture Example

Goal: Find the nearest Combini 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:

Rewards:

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

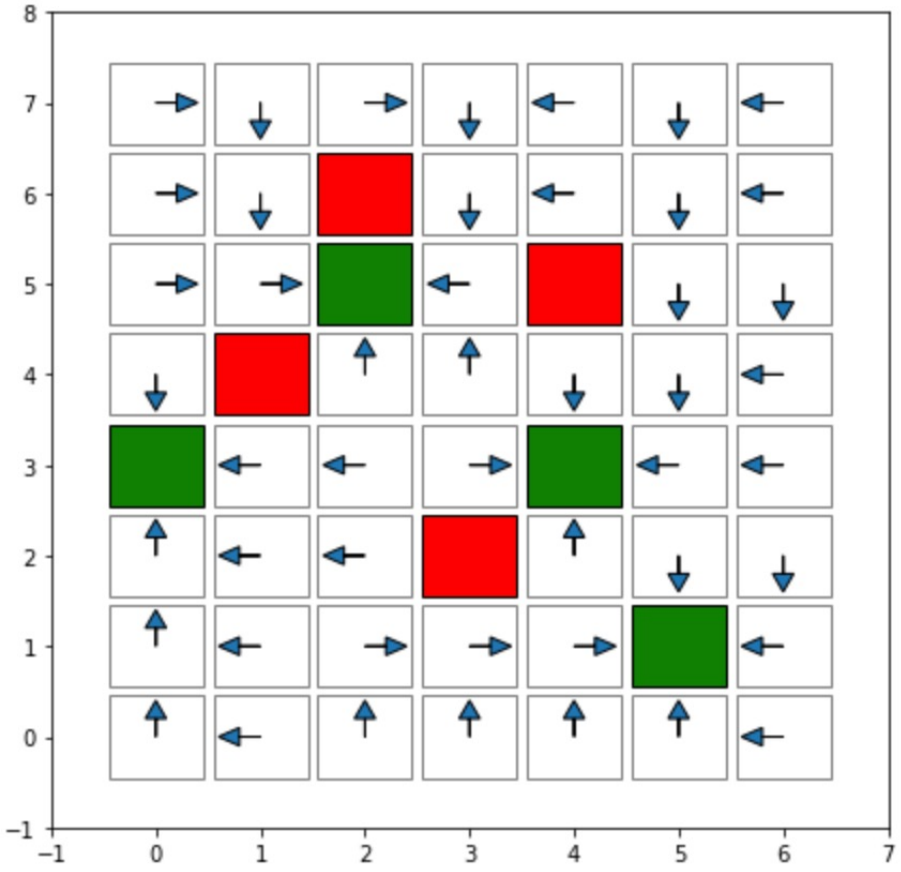
Lecture hall

Combini

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

Combini & Lecture Example: Possible optimal policy

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



Lecture hall

Combini

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