

MATHEMATICS FOR MACHINE LEARNING

Nagoya University, Fall 2023

Lecture 8

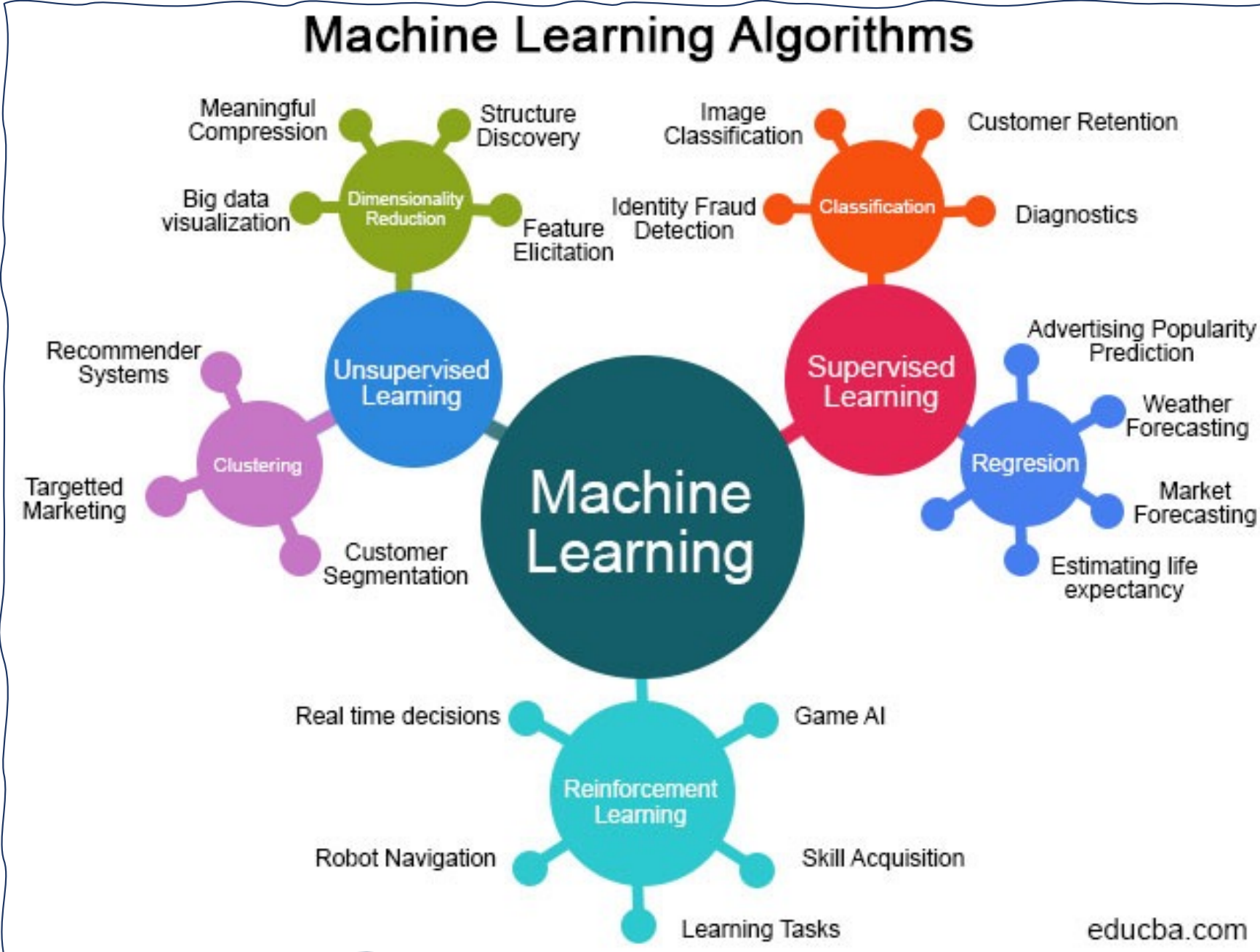
Reinforcement Learning: Q-Learning

This week Tutorial: ~~Wednesday 29th Nov. 5th period~~

Thursday 30th Nov 6th

<https://www.henrikbachmann.com/mml2023.html>

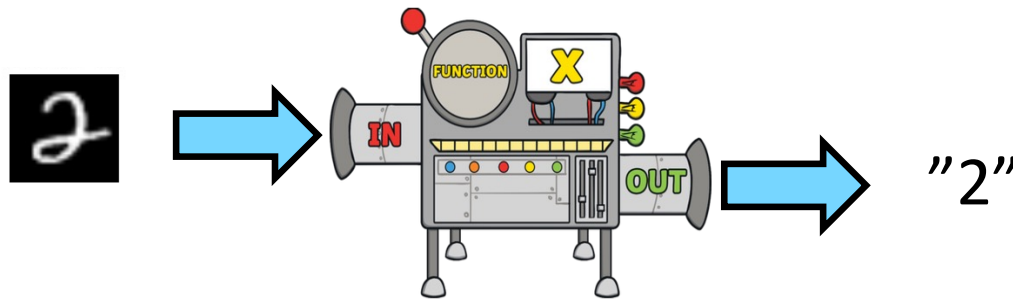
Overview



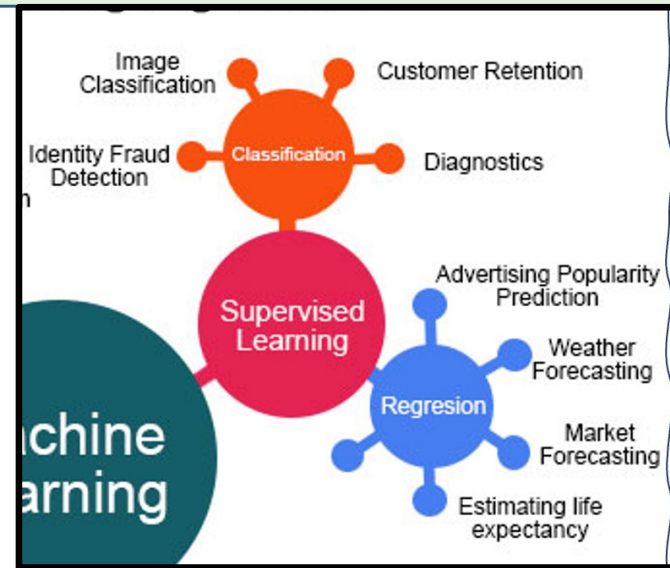
Supervised learning

Classification (discrete output)

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

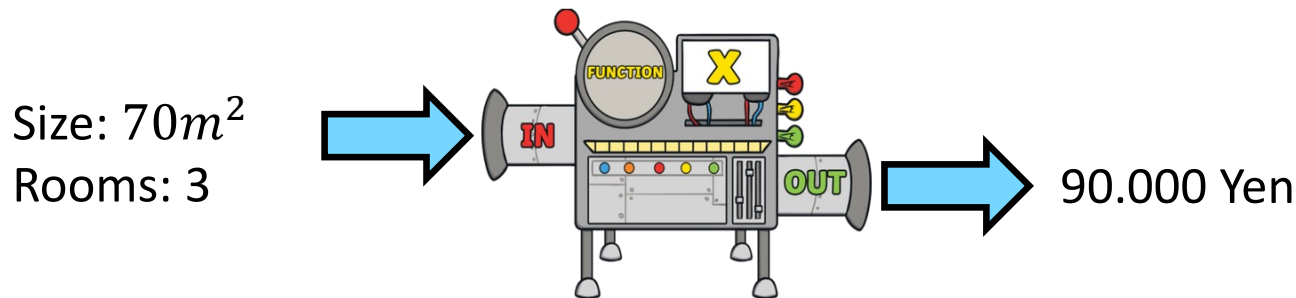


Handwriting recognition



Regression (continuous output)

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

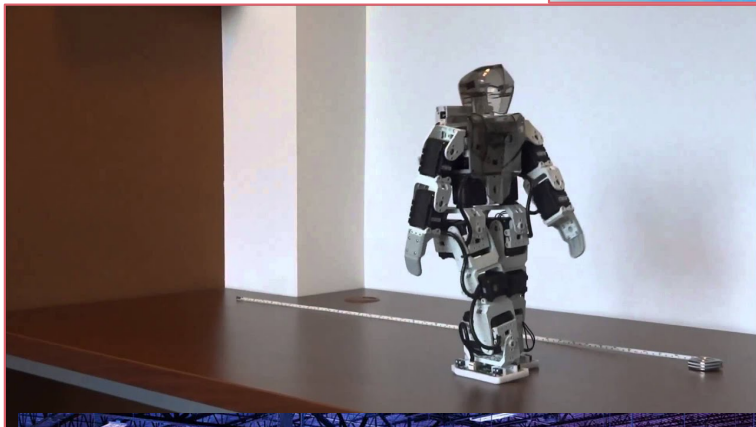
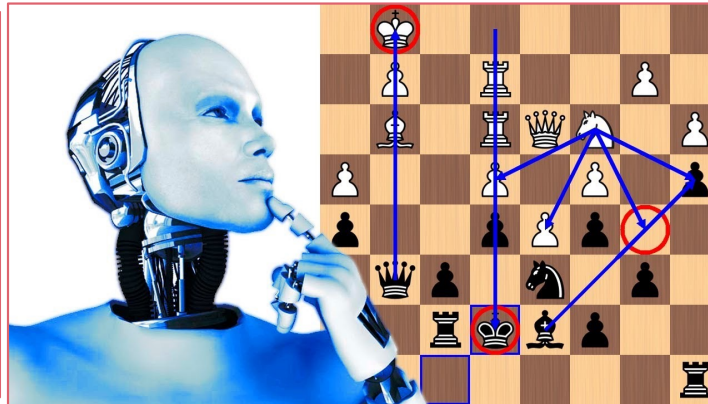


Apartment price prediction

Examples in this lecture so far

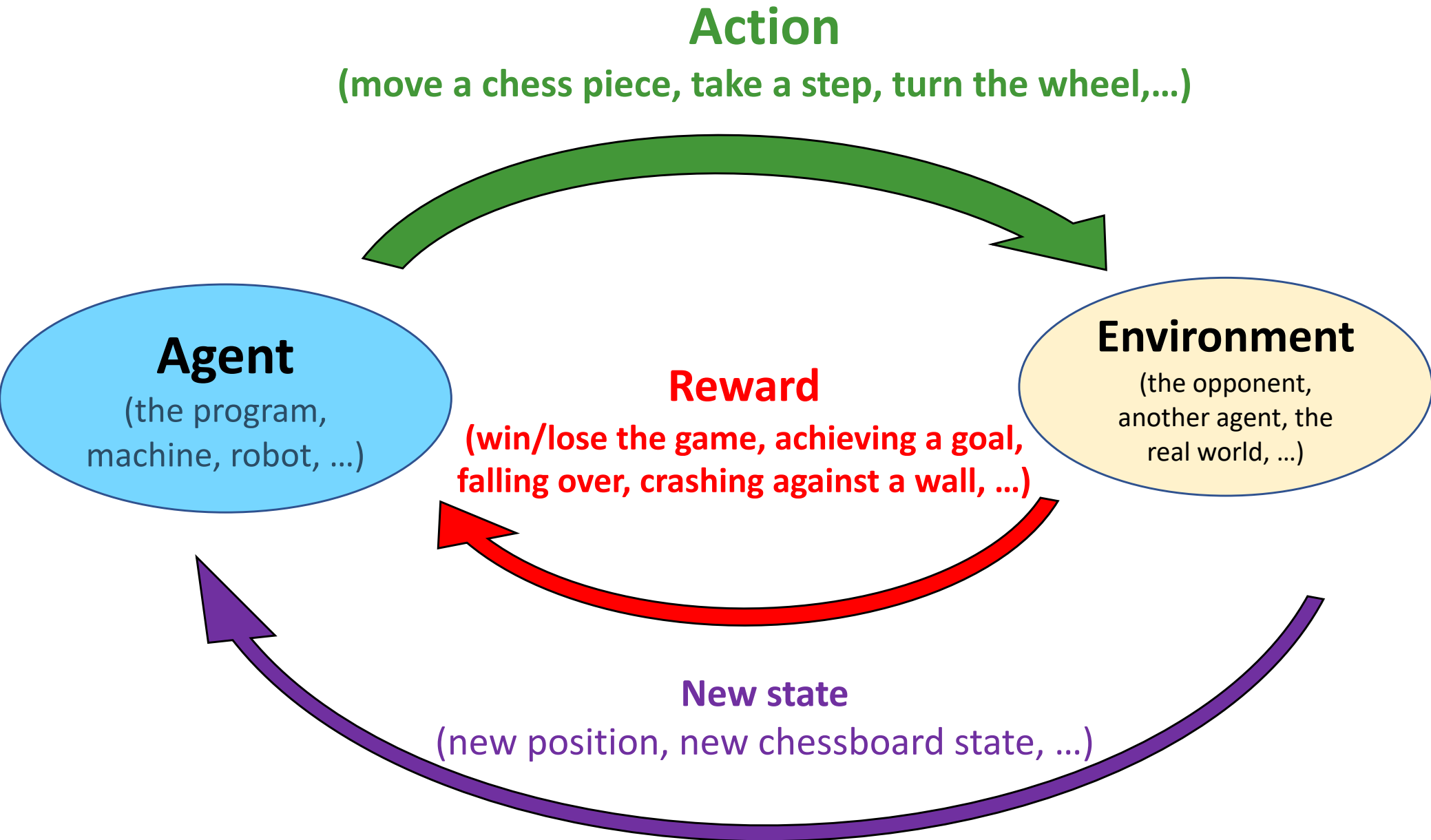
- Linear regression (Tebasaki)
- Logistic regression (Passing exam)
- Neural Network (digit recognition)

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>

Another example: https://www.youtube.com/watch?v=L_4BPjLBF4E

Markov decision process

Definition 5.1. 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

Definition 5.1. 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**.

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

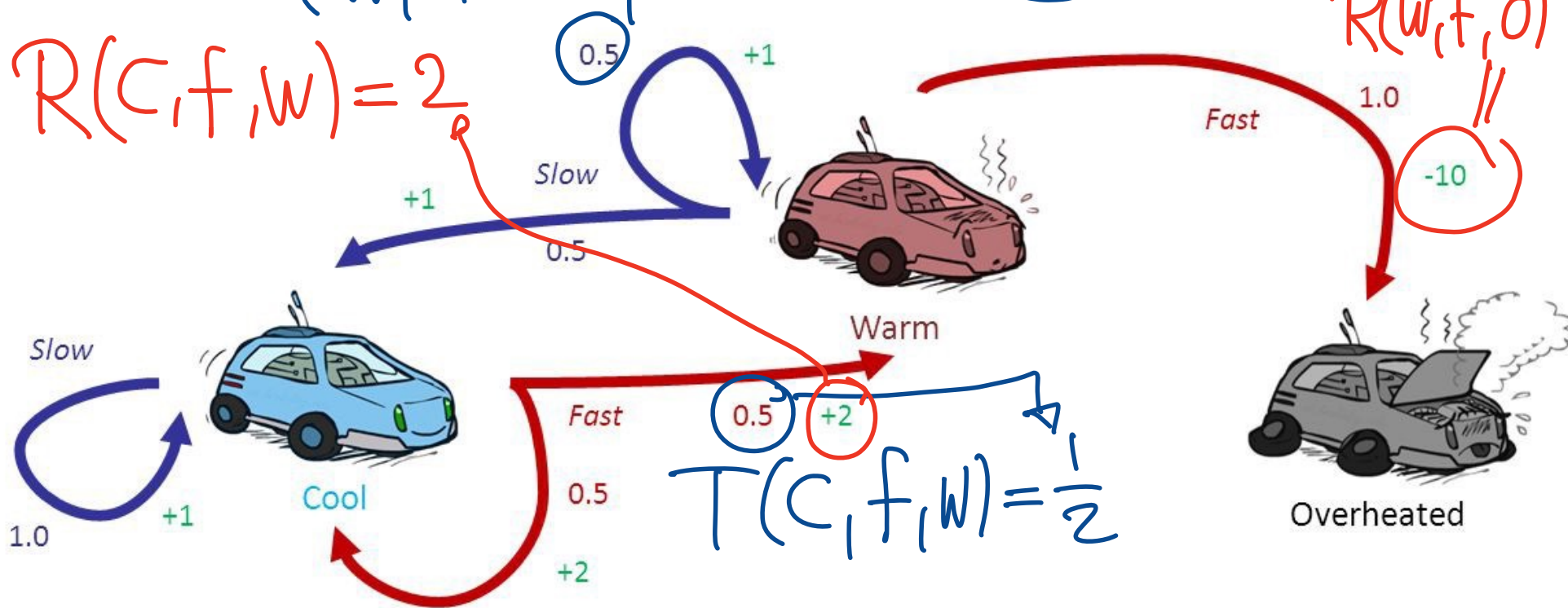
$$S = \{C_{\text{cool}}, W_{\text{warm}}, O_{\text{overheated}}\}$$

$$A = \{s, f\}$$

$$T(W, s, C) = \frac{1}{2} \quad T(W, s, W) = \frac{1}{2} \quad T(W, f, O) = 1$$

$$R(C, f, W) = 2$$

$$R(W, f, O) = -10$$



$$T(C, f, W) = \frac{1}{2}$$

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, \dots 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, \dots 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:

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

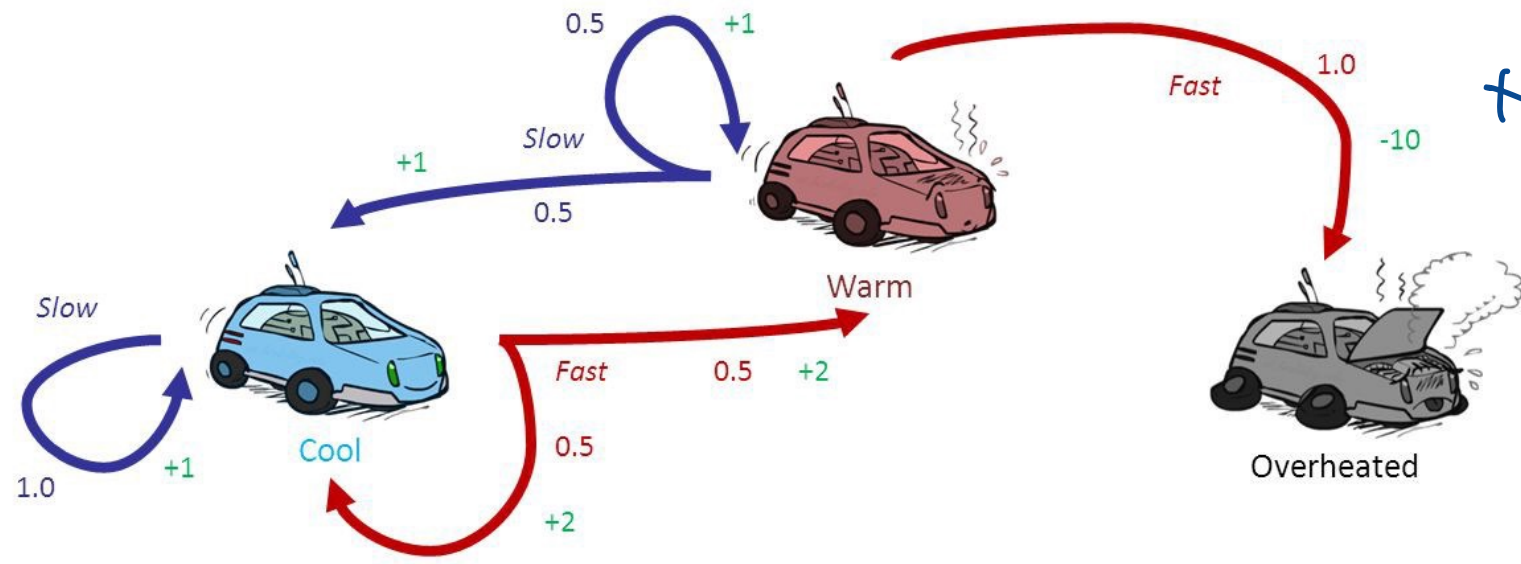
Another interpretation: Immediate rewards count more than delayed rewards.

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

$$\gamma = 0.1$$

$$\sum_{j \geq 0} \gamma^j R(s_j, a_j, s_{j+1}) = \underbrace{R(C, f, C)}_2 + \underbrace{(0.1) R(C, f, W)}_{0.2}$$



$$+ (0.1)^2 R(W, f, O)$$

$$\underbrace{-10}_{-0.1}$$

$$= 2.1$$

$$(s_0, s_1, s_2, \dots) = (C, C, W, O)$$

$$(a_0, a_1, a_2, \dots) = (f, f, f, \dots)$$

Markov decision process: Policy

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

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

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

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

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

Markov decision process: Value of a policy

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

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 $s \in S$:

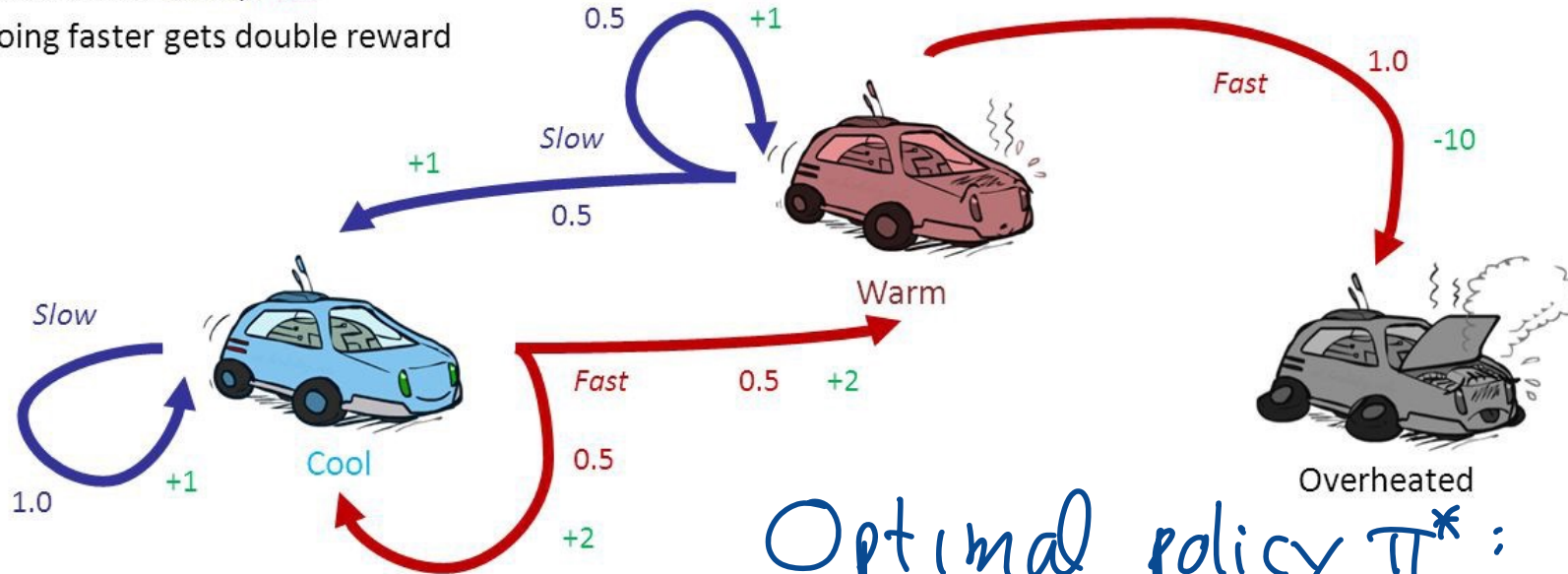
$$V_{\pi^*}(s) = \max_{\pi} V_{\pi}(s).$$

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



$$\pi^*(C) = F$$

$$\pi^*(W) = S$$

$$\pi^*(O) = S$$

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 .

$$Q^*(s, a) \equiv$$

In car example:

$$Q^*(C, s) = \underbrace{T(C, s, c)}_1 \underbrace{R(C, s, c)}_1 + \underbrace{T(C, s, w)}_0 \underbrace{R(C, s, w)}_0 + \underbrace{T(C, s, o)}_0 \underbrace{R(C, s, o)}_0$$

$$+ \gamma \frac{T(c, \rho, c)}{1} \cdot V_{\pi^*}(c)$$

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

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

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

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

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

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

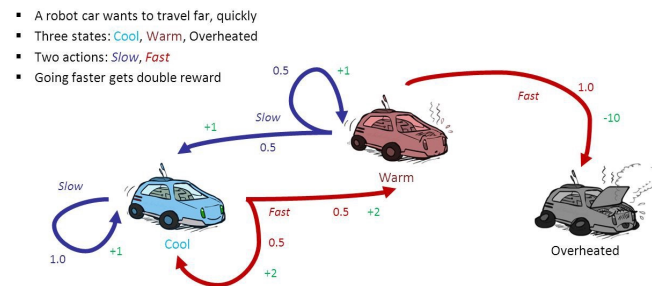
Goal rephrased: Find a good approximation of the state-action value function. “Learn” the function Q .

Q-learning: Basic idea & Car example

In the car Example:

$S = \{C, W, O\}$

$A = \{f, s\}$



The (approximated) state-action value function Q can be thought of as a table

		State		
		C (Cool)	W (warm)	O (overheat)
Action	f (fast)	? $Q(C, f)$? $Q(W, f)$? $Q(O, f)$
	s (slow)	? $Q(C, s)$? $Q(W, s)$? $Q(O, s)$

where each entry tells us the expected (best) total reward when taking an action in a certain state.

Q-learning: Algorithm to learn the entries of this table

Q-learning: Basic idea & Car example

Step 1: Start with some values, e.g. choose **random values** (or all 0):

		State		
		C (Cool)	W (warm)	O (overheat)
Action	f (fast)	3	2	0
	s (slow)	1	4	0

Q-learning: Basic idea & Car example

Step 1: Start with some values, e.g. choose **random values** (or all 0):

		State		
		C (Cool)	W (warm)	O (overheat)
Action	f (fast)	3	2	0
	s (slow)	1	4	0

Step 2: Choose a starting state, e.g. $s_0 = C$.


Q-learning: Basic idea & Car example

Step 0: Start with some values, e.g. choose **random values** (or all 0):

		State		
		C (Cool)	W (warm)	O (overheat)
Action	f (fast)	3	2	0
	s (slow)	1	4	0

Step 1: Choose a starting state, e.g. $s_0 = C$.

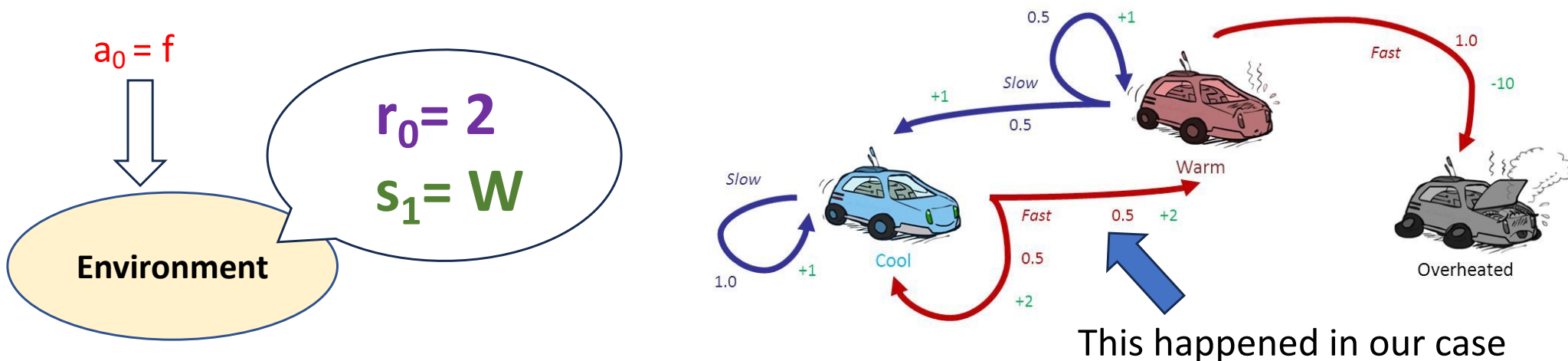
Step 2: Choose the current best action according to the table, i.e. find the **maximal entry** and set the first action a_0 to the action of the corresponding row.



		C (Cool)	W (warm)	O (overheat)
		$a_0 = f$	3	2
	s (slow)	1	4	0

Q-learning: Basic idea & Car example

Step 3: Take action $a_0 = f$ and receive a **reward** r_0 and a **new state** s_1 from the environment. For example



Question: How could we update the value $Q(C, f)$ now?

		$s_0 = C$	$s_1 = W$	
		C (Cool)	W (warm)	O (overheat)
$a_0 = f$	f (fast)	3	2	0
	s (slow)	1	4	0

Q-learning: Basic idea & Car example

		$s_0 = C$	$s_1 = W$	
		C (Cool)	W (warm)	O (overheat)
$a_0 = f$	f (fast)	3	2	0
	s (slow)	1	4	0

$r_0 = 2$

Step 4: Update $Q(C,f)$

We know that $Q(C,f)=3$ is random and might not be a good value.

But in later steps, we should assume that this is the currently best possible approximation. We should therefore also not overwrite it completely.

Idea:

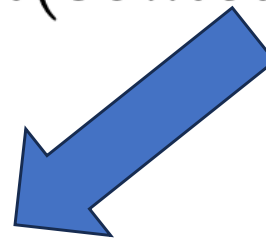
$$Q(C, f) = (1 - \alpha)Q(C, f) + \alpha(\text{something new...})$$

new value old value some number based on what just happened

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

Q-learning: Basic idea & Car example

$$Q(C, f) = (1 - \alpha)Q(C, f) + \alpha(\textit{something new...})$$



Should give an approximation of $Q(C, f)$ just based on what happened:

- We received a reward $r_0 = 2$
- We ended in state $s_1 = W$

Question: Based on this information what would be a good guess for $Q(C, f)$? Assume that the other values in the table below are good approximations.

Recall: $Q(C, f)$ tells us the expected total reward we get when taking action f in state C (and afterwards following the best possible policy).

		$s_0 = C$	$s_1 = W$	
		C (Cool)	W (warm)	O (overheat)
$a_0 = f$	f (fast)	3	2	0
	s (slow)	1	4	0

Q-learning: Basic idea & Car example

$$Q(s_0, a_0) = (1 - \alpha)Q(s_0, a_0) + \alpha(r_0 + \gamma \max_{a \in A} Q(s_1, a))$$
$$= (1 - \alpha)3 + \alpha(2 + \gamma 4).$$

$s_0 = C$ $s_1 = W$

	C (Cool)	W (warm)	O (overheat)
$a_0 = f$ f (fast)	3	2	0
s (slow)	1	4	0

For example, if $\alpha = \frac{1}{2}$ and $\gamma = \frac{3}{4}$ we get

$$Q(C, f) = \frac{3}{2} + \frac{1}{2} \left(2 + \frac{3}{4} 4 \right) = 4.$$

And the new table

	C (Cool)	W (warm)	O (overheat)
f (fast)	4	2	0
s (slow)	1	4	0

From here continue with Step 2...

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. $a_0 = \operatorname{argmax}_{a \in A} Q(s_0, a)$.
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!

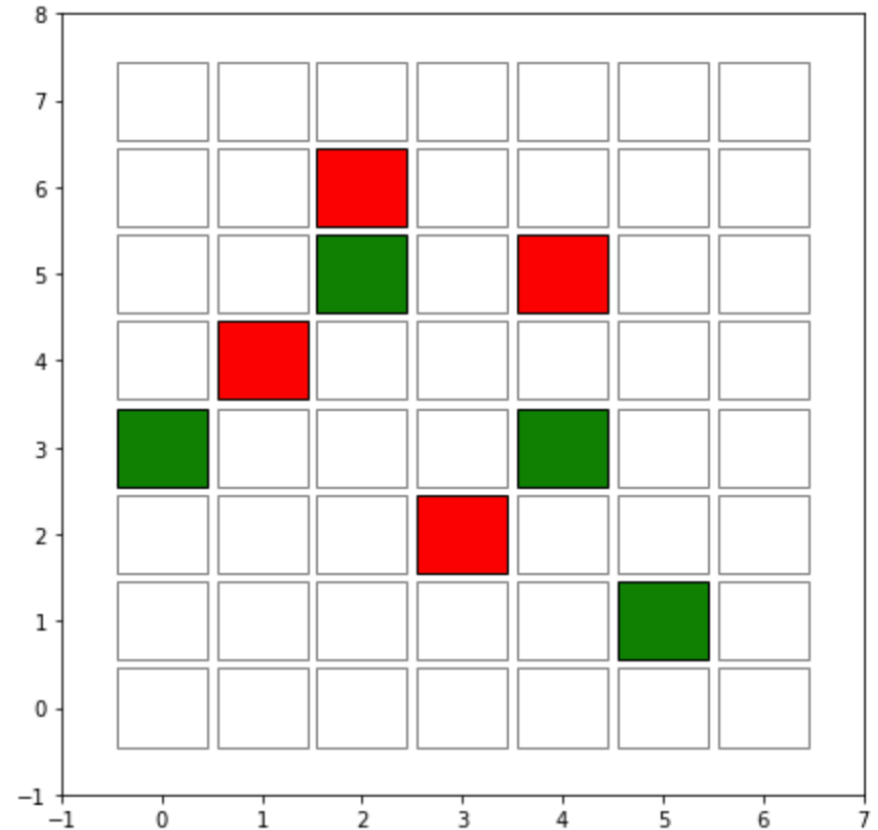
States: $7 * 8 = 56$ positions
(terminal states:  )

Actions:    

Rewards:

- 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

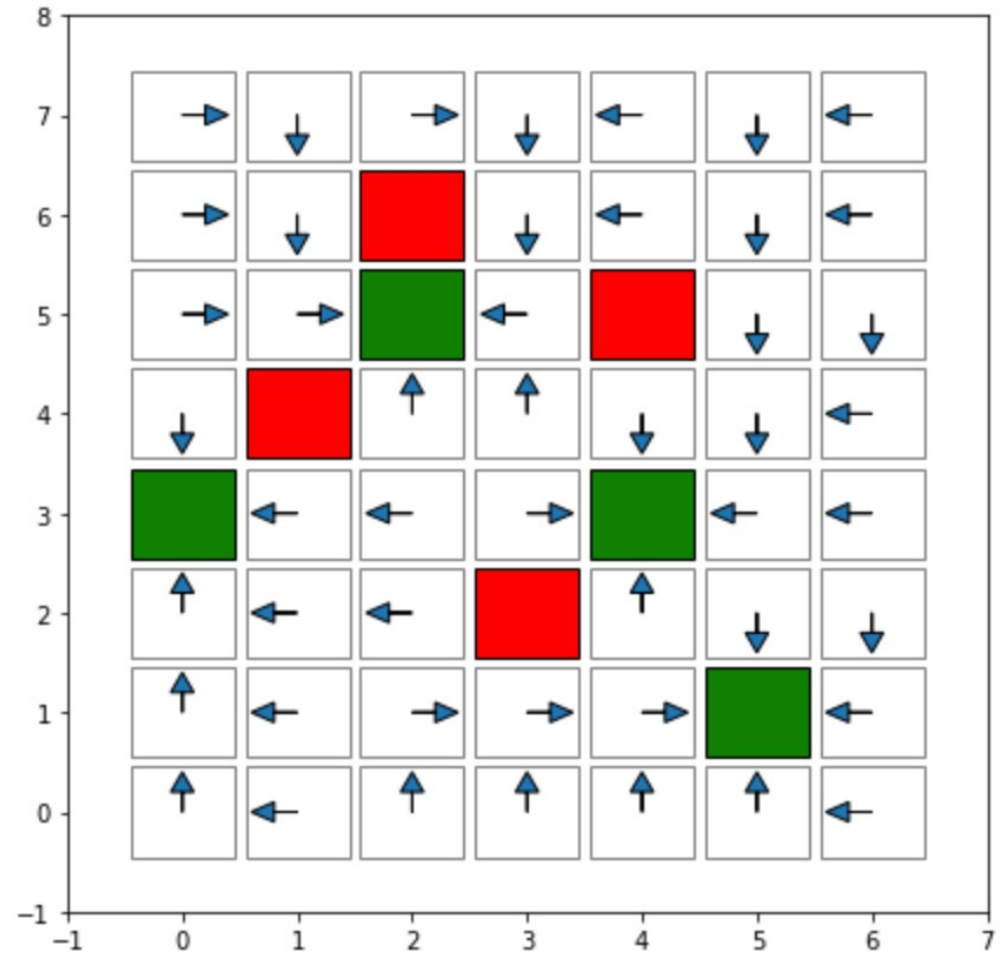


 **Lecture hall**
 **Comбини**

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**