Fundamentals of Artificial Intelligence – Learning

Matthias Althoff

TU München

Winter semester 2022/23

Organization

1. Types of Learning
2. Supervised Learning
3. Learning Decision Trees
4. Reinforcement Learning
   - Case Study: Safe RL for Autonomous Lane Changing
   - Case Study: Safe RL for Human–Robot Collaboration
   - Current Research: Learning for Power Systems

The content is covered in the AI book by the sections “Learning from Examples” and “Reinforcement Learning”.

Learning Outcomes

- You can explain the difference between *unsupervised learning*, *reinforcement learning*, and *supervised learning*.
- You understand that *hypothesis selection* is crucial for *supervised learning*.
- You know the definition of a *decision tree*.
- Given a *decision tree*, you can explain how a decision based on that tree is made.
- You can explain why a *decision tree* can express any function of input attributes.
- You can apply the proposed *decision tree learning algorithm*.
- You can quantify the importance of attributes using *information-theoretic entropy*.
- You can explain basic *reinforcement learning* methods.

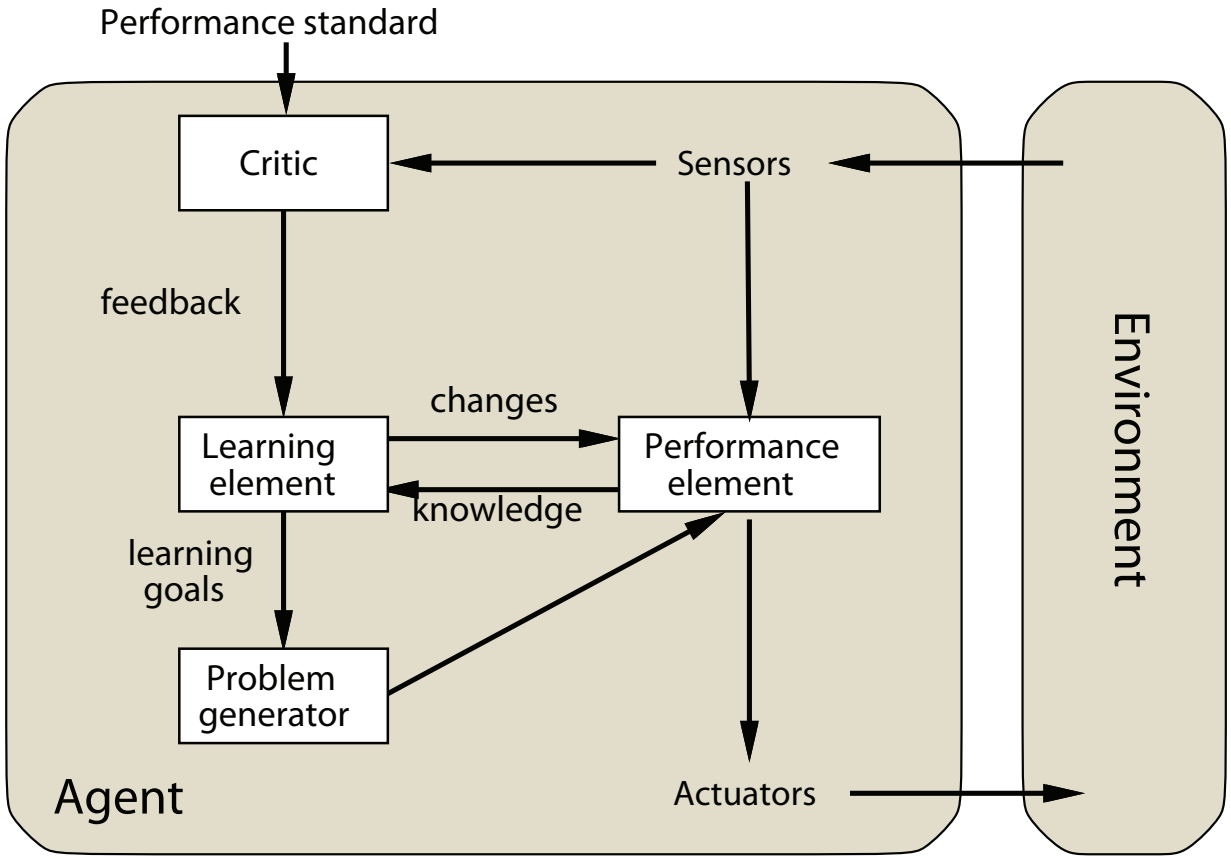
Introduction

An agent is **learning** if it improves its performance on future tasks after making observations about the world.

Motivation for learning

- Designers cannot anticipate all possible situations that an agent might face.
- Designers cannot anticipate all changes over time (e.g., stock market prediction).
- Human programmers might not have an idea how to program a solution themselves (e.g., face recognition).

Reminder: Learning Agent



- Any previous agent can be extended to a learning agent.
- The block **Performance element** is a placeholder for the whole of any of the previous agents.

Types of Learning

Unsupervised learning

- The agent learns patterns in the input even though no feedback is supplied.
- Most common technique is clustering, e.g., automated car might develop a concept for good and bad traffic without providing labeled examples.

Reinforcement learning (presented in this lecture)

- The agent learns from a series of reinforcements – rewards or punishments.
- For instance, winning a chess game tells the agent it did something right.

Supervised learning (focus of this lecture)

- The agent observes some example input-output pairs and learns a function that maps from input to output.
- For instance, inputs for automated driving are sensor values, and outputs are instructions from a human (e.g., "brake").

Supervised Learning

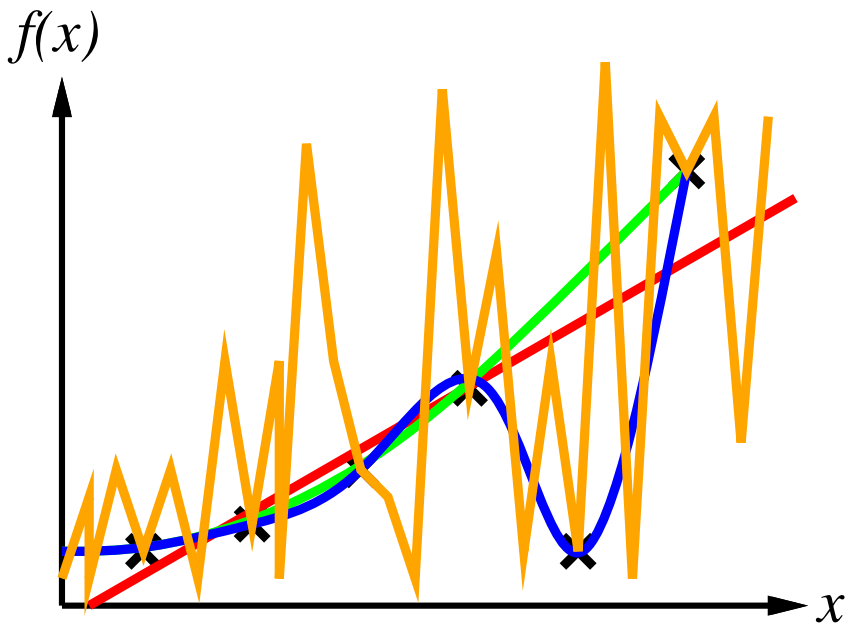
Task of supervised learning

Given a **training set** of N example input-output pairs

$$(x_1, y_1), (x_2, y_2), \ldots (x_N, y_N),$$

where each y_i was generated by an unknown function $y = f(x)$, discover a function h that approximates f as well as possible.

E.g., curve fitting:



Ockham's razor: choose the simplest consistent hypothesis

Hypothesis Selection

Finding a good hypothesis $h(x) \in \mathcal{H}$ out of the set of possible hypothesis functions $\mathcal{H}$ is crucial. We wish to maximize the probability that a hypothesis belongs to a data set:

$$h^* = \arg\max_{h \in \mathcal{H}} P(h|data)$$

using Bayes' rule we obtain

$$h^* = \arg\max_{h \in \mathcal{H}} P(data|h)P(h).$$

We can say that simple hypotheses have high probability, e.g., the previous data points belong to a degree-1 or -2 polynomial and thus prevent the spiky solution.

Generalization

We use a **test set** to check whether the hypothesis $h(x)$ **generalizes** well and predicts other values outside the training set (test set $\neq$ training set).

Decision Trees

Unfortunately, decision trees in decision tree learning are differently defined compared to the lecture *Rational Decisions* for historical reasons:

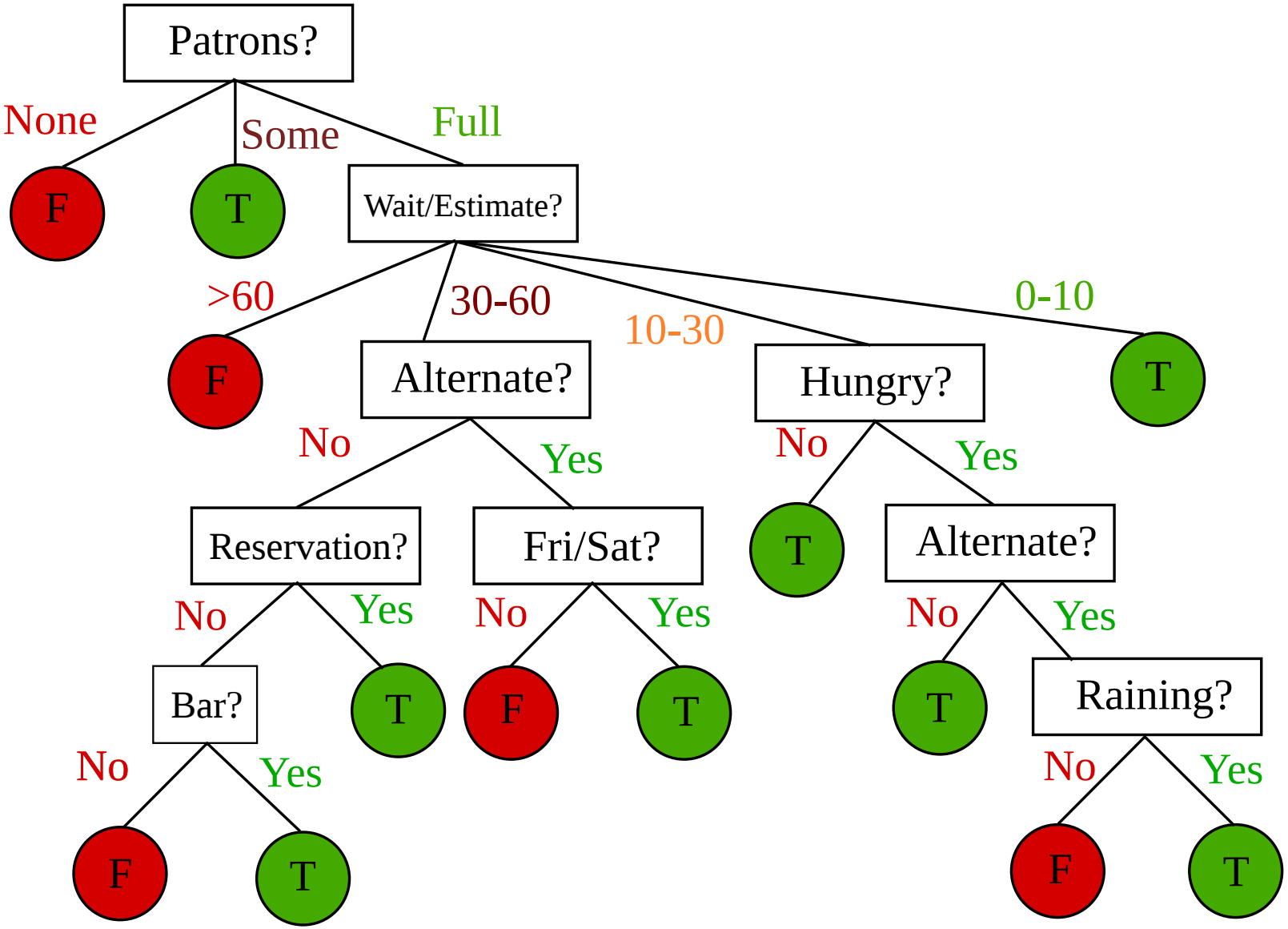
- A **decision tree** represents a function that takes a vector of attribute values as input and returns a "decision" – a single Boolean output value.
- We restrict ourselves to discrete inputs.
- A decision tree reaches its decision through a sequence of tests:
  - An internal node represents a test of a property;
  - Edges are annotated with the possible test values;
  - Each leaf node has the Boolean value which should be returned.
- Decision trees are natural for humans, e.g., "How To" manuals (e.g., for car repair) are often written in a decision tree structure.

Example: Restaurant (DecisionTreeLearning.ipynb)

Goal predicate: *WillWait*

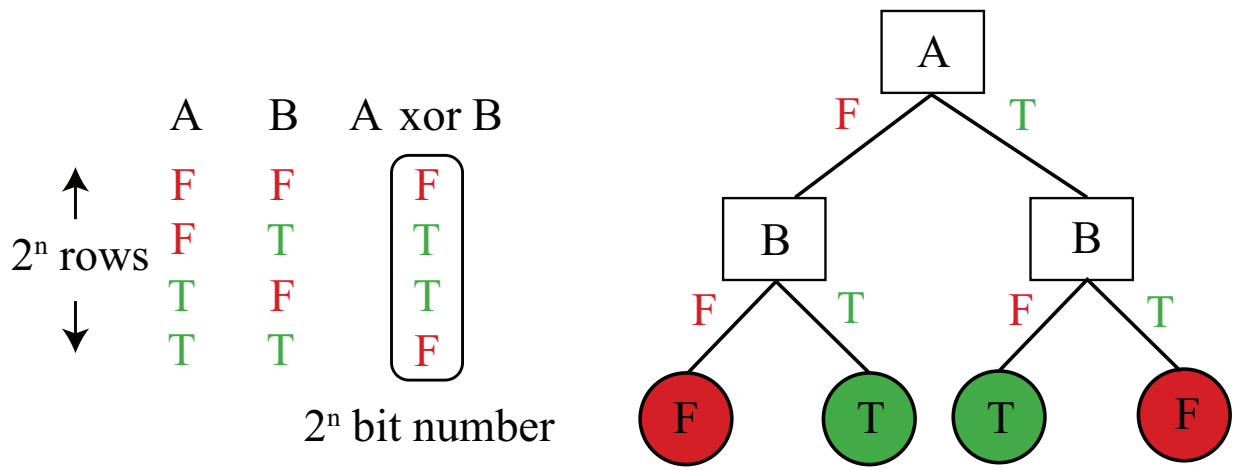
Patrons	:	How many guests? (none, some, full)
WaitEstimate	:	How long is the waiting time? ($0-10$, $10-30$, $30-60$, >60)
Alternate	:	Are there alternatives? (True/False)
Hungry	:	Am I hungry? (T/F)
Reservation	:	Do I have a reservation? (T/F)
Bar	:	Is there a bar where I can wait for my seat? (T/F)
Fri/Sat	:	Is it Friday or Saturday? (T/F)
Raining	:	Is it raining outside? (T/F)
Price	:	How high are the prices? (\$, \$\$, \$\$\$)
Type	:	Type of restaurant (French, Italian, Thai, burger)

Decision Tree for the Restaurant (DecisionTreeLearning.ipynb)



Expressiveness

Decision trees can express any function of the input attributes.
E.g., for Boolean functions, truth table row → path to leaf:



- There is a consistent decision tree for any training set with one path to a leaf for each example (unless f is nondeterministic in x), but it probably will not generalize to new examples.
- 2^n rows for n Boolean attributes results in 2^n bit number; n bits represent 2^n different numbers → 2^{2^n} possible decision trees.
- We prefer to find more *compact* decision trees.

Training Set for the Decision Tree (DecisionTreeLearning.ipynb)

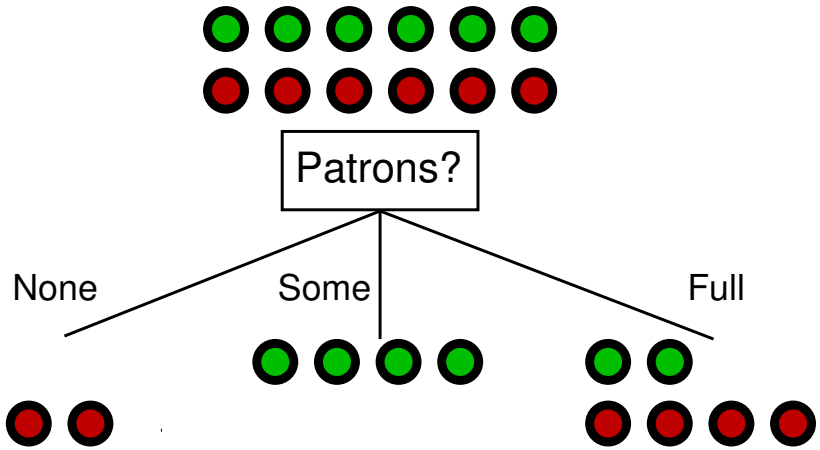
Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	T	F	F	T	Some	$$$	F	T	French	0–10	T
X_2	T	F	F	T	Full	$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	$$$	F	T	French	>60	F
X_6	F	T	F	T	Some	$$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	$$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	$$$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	$	F	F	Burger	30–60	T

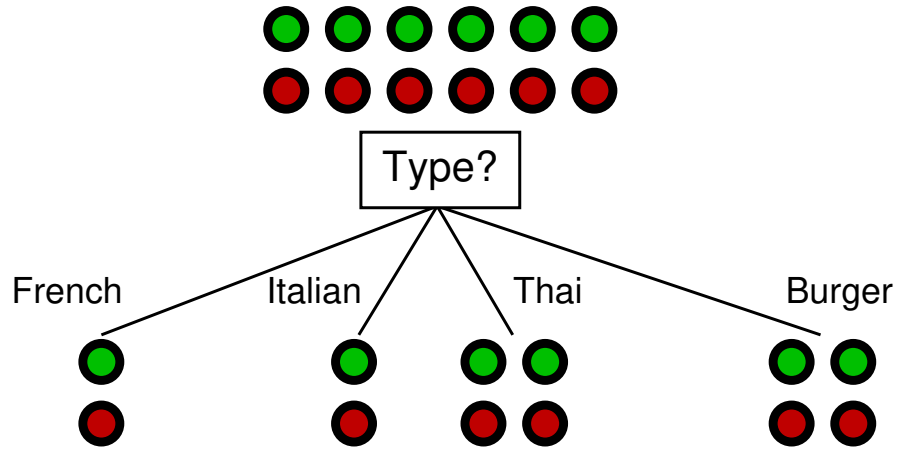
Inducing Decision Trees from Examples

- It is intractable to find the smallest consistent tree from the training set (2^{2^n} possible decision trees).
- We need efficient heuristics!
- The `Decision-Tree-Learning` algorithm adopts a greedy divide-and-conquer strategy:
  1. Test the most important attribute first.
  2. The test divides the problem into two smaller subproblems that can be solved recursively.

Tweedback Question

What attribute would you choose?



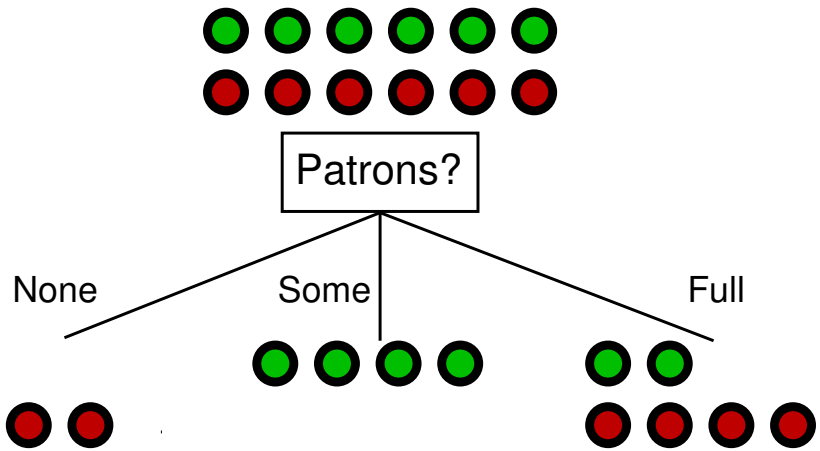


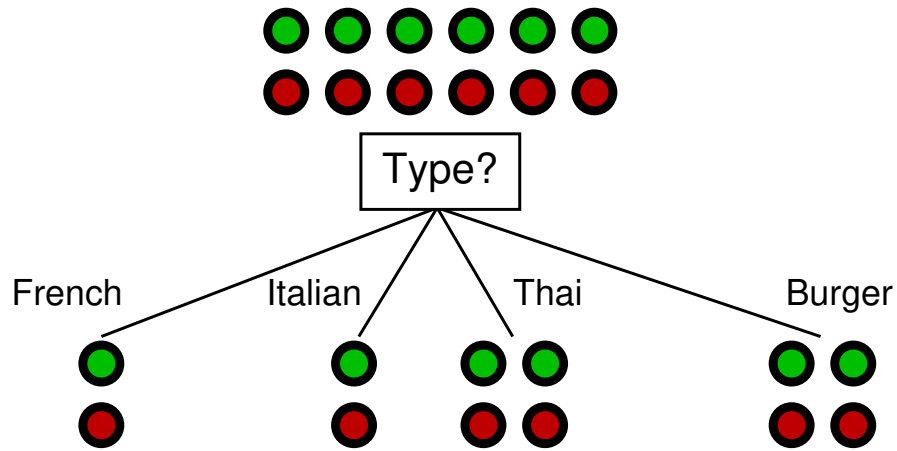
A Patrons

B Type

Choosing an Attribute

Idea: A good attribute splits the examples into subsets that are (ideally) "all positive" or "all negative"





Patrons? is a better choice – gives *information* about the classification

Quantifying Importance of Attributes (1)

We will use entropy (not in the thermodynamic sense) for finding a good attribute to split the examples:

Information-theoretic entropy

Entropy is a measure of the uncertainty of a random variable; acquisition of information corresponds to a reduction in entropy.

The entropy of a random variable V with values v_k is defined as

$$H(V) = \sum_k P(v_k) \log_2 \frac{1}{P(v_k)} = -\sum_k P(v_k) \log_2 P(v_k).$$

Examples:

- Fair coin: $H(Fair) = -(0.5 \log_2 0.5 + 0.5 \log_2 0.5) = 1$.
- Loaded coin (99% heads):
  $H(Loaded) = -(0.99 \log_2 0.99 + 0.01 \log_2 0.01) \approx 0.08$.

The unit of entropy is *bit*: $H(Fair) = 1$ bit, $H(Loaded) \approx 0.08$ bit.

Quantifying Importance of Attributes (2)

We introduce $B(q)$ as the entropy of a Boolean random variable that is true with probability q:

$$B(q) = -(q \log_2 q + (1-q) \log_2 (1-q)).$$

If a training set contains p positive examples and n negative examples, the entropy of the goal attribute on the whole set is

$$H(goal) = B\left(\frac{p}{p+n}\right).$$

The restaurant training set on slide 13 has $p = n = 6$ so that $H(Goal) = B(0.5) = 1$.

Quantifying Importance of Attributes (3)

- An attribute A with d values divides the training set E into subsets $E_1, \ldots, E_d$.
- Each subset E_k has p_k positive and n_k negative examples, resulting in $B(p_k/(p_k + n_k))$ bits of information to answer the question.
- A randomly chosen example from the training set has the kth value with probability $(p_k + n_k)/(p + n)$, so the expected entropy remaining is

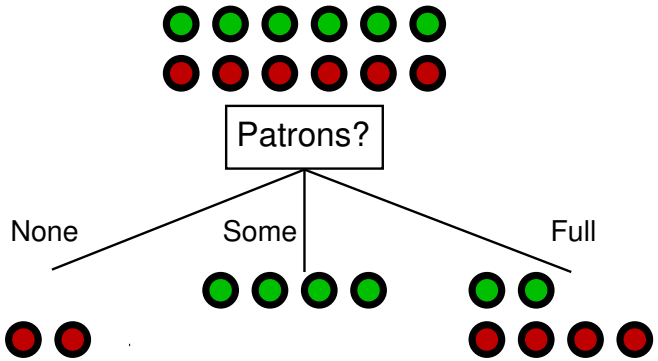
$$Remainder(A) = \sum_{k=1}^{d} \frac{p_k + n_k}{p + n} B\left(\frac{p_k}{p_k + n_k}\right).$$

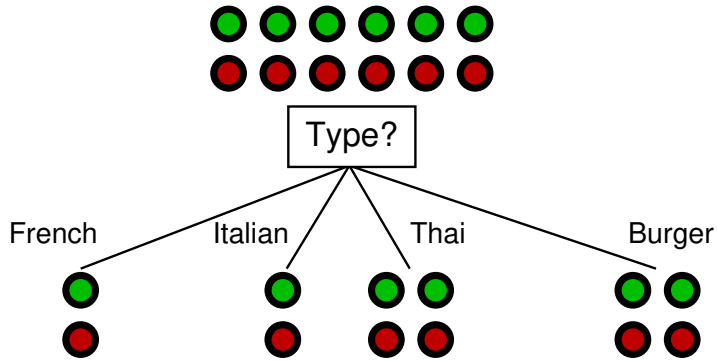
- The **information gain** from the attribute test on A is the expected reduction in entropy:

$$Gain(A) = B\left(\frac{p}{p + n}\right) - Remainder(A).$$

Quantifying Importance of Attributes (4)

Examples:





- $Gain(Patrons) = 1 - [\frac{2}{12}B(\frac{0}{2}) + \frac{4}{12}B(\frac{4}{4}) + \frac{6}{12}B(\frac{2}{6})] \approx 0.541.$
- $Gain(Type) = 1 - [\frac{2}{12}B(\frac{1}{2}) + \frac{2}{12}B(\frac{1}{2}) + \frac{4}{12}B(\frac{2}{4}) + \frac{4}{12}B(\frac{2}{4})] = 0.$

This confirms that *Patrons* is a better attribute to split on.

Four Cases in Attribute Selection

1. **Remaining examples are all positive (or all negative):**
   we can answer with *Yes* or *No* and are done.
2. **There are some positive and some negative examples:**
   choose the best attribute to split them using $Gain(A)$ on slide 19.
3. **There are no examples left (no example for this combination of attribute values):**
   return a default value calculated from the plurality classification of all the examples that were used in constructing the node's parent (i.e., selects the most common output value among examples).
4. **There are no attributes left, but both positive and negative examples:**
   there is a conflict. Reasons:
   - noise in the data;
   - domain is nondeterministic;
   - we cannot observe an attribute that would distinguish the examples.

   Best option: return plurality classification of the remaining examples.

Decision Tree Learning Algorithm (DecisionTreeLearning.ipynb)

function DecisionTreeLearning(*examples*, *attributes*, *parent_examples*) **returns** tree

if *examples* is empty **then return** `Plurality-Value`(*parent_examples*)
else if all *examples* have the same classification **then return** the classification
else if *attributes* is empty **then return** `Plurality-Value`(*examples*)
else
  $A \leftarrow \arg\max_{a \in attributes}$ `Importance`$(a, examples)$
  tree $\leftarrow$ a new decision tree with root test A
  for each value v_k of A **do**
    $exs \leftarrow \{e | e \in examples \text{ and } e.A = v_k\}$
    subtree $\leftarrow$ `DecisionTreeLearning`$(exs, attributes \setminus A, examples)$
    add a branch to *tree* with label $(A = v_k)$ and subtree *subtree*
  return *tree*

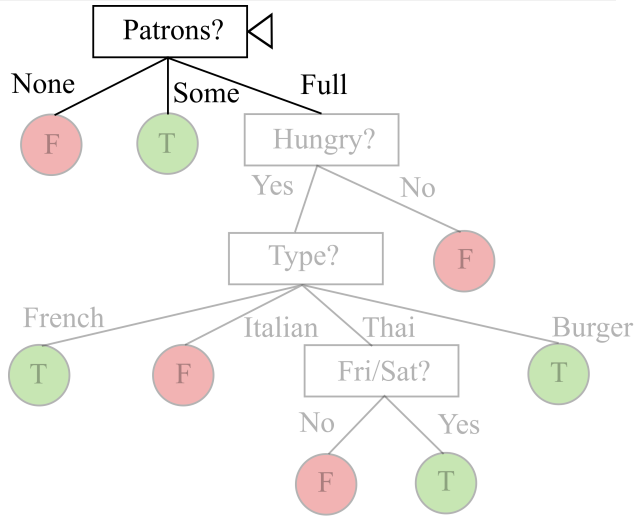
A: input attribute;
v_k: possible value of input attribute A;
`Plurality-Value`: selects the most common output value among examples.

Decision Tree Learning Algorithm (DecisionTreeLearning.ipynb)

function DecisionTreeLearning (*examples*, *attributes*, *parent_examples*) **returns** tree

if *examples* is empty **then return** `Plurality-Value`(*parent_examples*)
else if all *examples* have the same classification **then return** the classification
else if *attributes* is empty **then return** `Plurality-Value`(*examples*)
else
  $A \leftarrow \arg\max_{a \in attributes}$ `Importance`$(a, examples)$
  tree $\leftarrow$ a new decision tree with root test A
  for each value v_k of A **do**
    $exs \leftarrow \{e | e \in examples$ **and** $e.A = v_k\}$
    subtree $\leftarrow$ `DecisionTreeLearning`$(exs, attributes \setminus A, examples)$
    add a branch to *tree* with label $(A = v_k)$ and subtree *subtree*
  return *tree*

examples:
$\{X_1 = (T, \ldots, 0-10, T), \ldots, X_{12} = (T, \ldots, 30-60, T)\}$
attributes: [Alt, Bar, Fri, Hun, Pat, Price, Rain, Res, Type, Est]
parent_examples: $\{X_1, \ldots, X_{12}\}$
A: "Patrons"



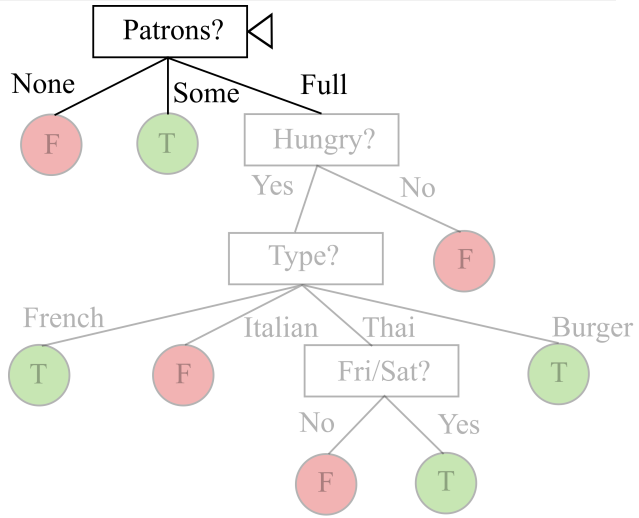
Decision Tree Learning Algorithm (DecisionTreeLearning.ipynb)

function DecisionTreeLearning (*examples*, *attributes*, *parent_examples*) **returns** tree

if *examples* is empty **then return** `Plurality-Value`(*parent_examples*)
else if all *examples* have the same classification **then return** the classification
else if *attributes* is empty **then return** `Plurality-Value`(*examples*)
else
  $A \leftarrow \arg\max_{a \in attributes}$ `Importance`$(a, examples)$
  tree $\leftarrow$ a new decision tree with root test A
  for each value v_k of A **do**
    $exs \leftarrow \{e | e \in examples \text{ and } e.A = v_k\}$
    subtree $\leftarrow$ `DecisionTreeLearning`$(exs, attributes \setminus A, examples)$
    add a branch to *tree* with label $(A = v_k)$ and subtree *subtree*
  return *tree*

examples: $\{X_1, \ldots, X_{12}\}$
attributes: [Alt, Bar, Fri, Hun, Pat, Price, Rain, Res, Type, Est]
parent_examples: $\{X_1, \ldots, X_{12}\}$
A: "Patrons" v_k: "None"

	Example	Pat	...	WillWait
exs:	X_7	None	...	F
	X_{11}	None	...	F



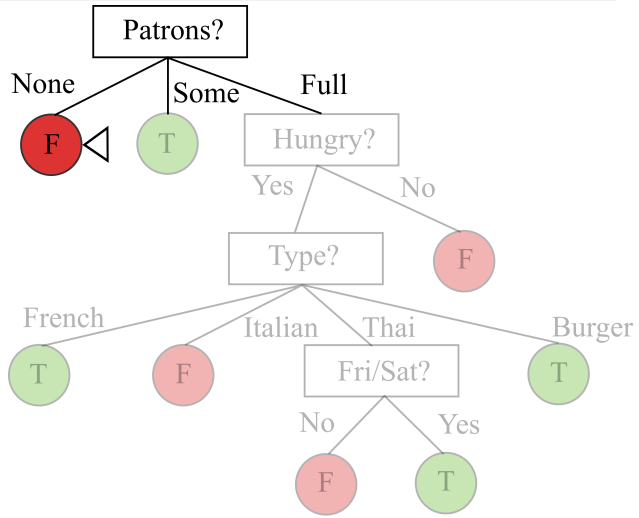
Decision Tree Learning Algorithm (DecisionTreeLearning.ipynb)

function DecisionTreeLearning (*examples*, *attributes*, *parent_examples*) **returns** tree

if *examples* is empty **then return** `Plurality-Value`(*parent_examples*)
else if all *examples* have the same classification **then return** the classification
else if *attributes* is empty **then return** `Plurality-Value`(*examples*)
else
  $A \leftarrow \arg\max_{a \in attributes}$ `Importance`$(a, examples)$
  tree $\leftarrow$ a new decision tree with root test A
  for each value v_k of A **do**
    $exs \leftarrow \{e | e \in examples$ **and** $e.A = v_k\}$
    subtree $\leftarrow$ `DecisionTreeLearning`$(exs, attributes \setminus A, examples)$
    add a branch to *tree* with label $(A = v_k)$ and subtree *subtree*
  return *tree*

	Example	Pat	...	WillWait
examples:	X_7	None	...	F
	X_{11}	None	...	F

attributes: [Alt, Bar, Fri, Hun, Price, Rain, Res, Type, Est]
parent_examples: $\{X_1, \ldots, X_{12}\}$



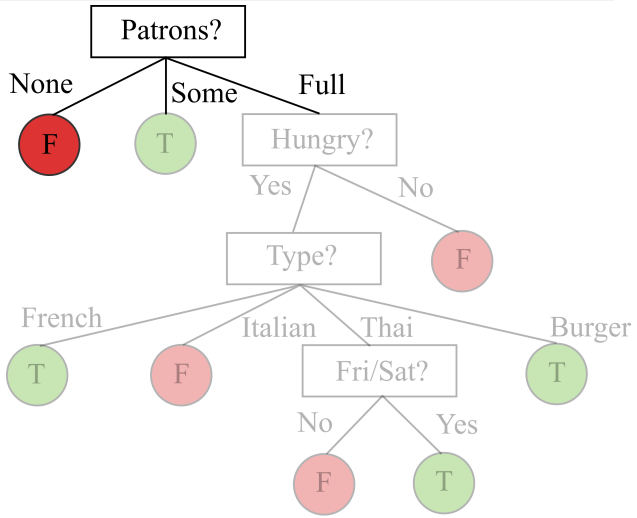
Decision Tree Learning Algorithm (DecisionTreeLearning.ipynb)

function DecisionTreeLearning (*examples*, *attributes*, *parent_examples*) **returns** tree

if *examples* is empty **then return** `Plurality-Value`(*parent_examples*)
else if all *examples* have the same classification **then return** the classification
else if *attributes* is empty **then return** `Plurality-Value`(*examples*)
else
  $A \leftarrow \arg\max_{a \in attributes}$ `Importance`$(a, examples)$
  tree $\leftarrow$ a new decision tree with root test A
  for each value v_k of A **do**
    exs $\leftarrow \{e | e \in examples$ **and** $e.A = v_k\}$
    subtree $\leftarrow$ `DecisionTreeLearning`(*exs*, *attributes* $\setminus$ A, *examples*)
    add a branch to *tree* with label $(A = v_k)$ and subtree *subtree*
  return *tree*

examples: $\{X_1, \ldots, X_{12}\}$
attributes: [Alt, Bar, Fri, Hun, Pat, Price, Rain, Res, Type, Est]
parent_examples: $\{X_1, \ldots, X_{12}\}$
A: "Patrons" v_k: "Some"

	Example	Pat	...	WillWait
	X_1	Some	...	T
exs:	X_3	Some	...	T
	X_6	Some	...	T
	X_8	Some	...	T



Decision Tree Learning Algorithm (DecisionTreeLearning.ipynb)

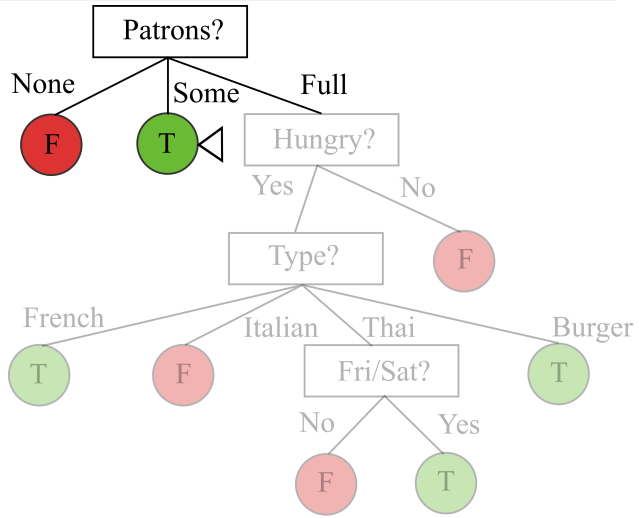
function DecisionTreeLearning (*examples*, *attributes*, *parent_examples*) **returns** tree

if *examples* is empty **then return** `Plurality-Value`(*parent_examples*)
else if all *examples* have the same classification **then return** the classification
else if *attributes* is empty **then return** `Plurality-Value`(*examples*)
else
  $A \leftarrow \arg\max_{a \in attributes}$ `Importance`$(a, examples)$
  tree $\leftarrow$ a new decision tree with root test A
  for each value v_k of A **do**
    $exs \leftarrow \{e | e \in examples$ **and** $e.A = v_k\}$
    subtree $\leftarrow$ `DecisionTreeLearning`$(exs, attributes \setminus A, examples)$
    add a branch to *tree* with label $(A = v_k)$ and subtree *subtree*
  return *tree*

examples:

Example	Pat	...	WillWait
X_1	Some	...	T
X_3	Some	...	T
X_6	Some	...	T
X_8	Some	...	T

attributes: [Alt, Bar, Fri, Hun, Price, Rain, Res, Type, Est]
parent_examples: $\{X_1, \ldots, X_{12}\}$



Decision Tree Learning Algorithm (DecisionTreeLearning.ipynb)

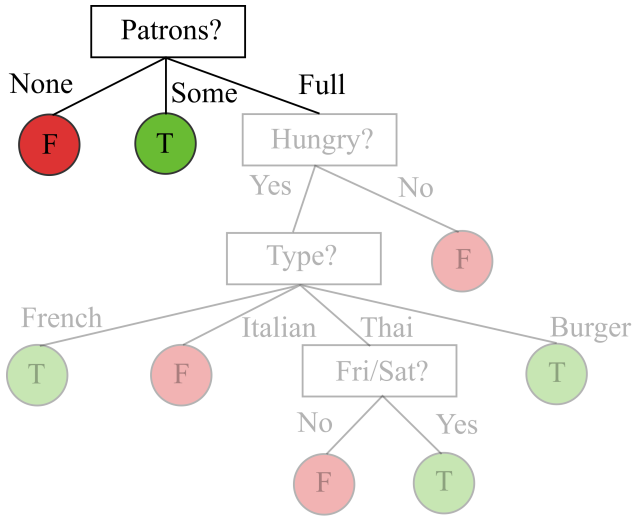
function DecisionTreeLearning (*examples*,*attributes*,*parent_examples*) **returns** tree

if *examples* is empty **then return** `Plurality-Value`(*parent_examples*)
else if all *examples* have the same classification **then return** the classification
else if *attributes* is empty **then return** `Plurality-Value`(*examples*)
else
  $A \leftarrow \arg\max_{a \in attributes}$ `Importance`$(a, examples)$
  tree $\leftarrow$ a new decision tree with root test A
  for each value v_k of A **do**
    $exs \leftarrow \{e | e \in examples \text{ and } e.A = v_k\}$
    subtree $\leftarrow$ `DecisionTreeLearning`$(exs, attributes \setminus A, examples)$
    add a branch to *tree* with label $(A = v_k)$ and subtree *subtree*
  return *tree*

examples: $\{X_1, \ldots, X_{12}\}$
attributes: [Alt, Bar, Fri, Hun, Pat, Price, Rain, Res, Type, Est]
parent_examples: $\{X_1, \ldots, X_{12}\}$ A: "Patrons" v_k: "Full"

exs:

Example	Pat	...	WillWait
X_2	Full	...	F
X_4	Full	...	T
X_5	Full	...	F
X_9	Full	...	F
X_{10}	Full	...	F
X_{12}	Full	...	T



Decision Tree Learning Algorithm (DecisionTreeLearning.ipynb)

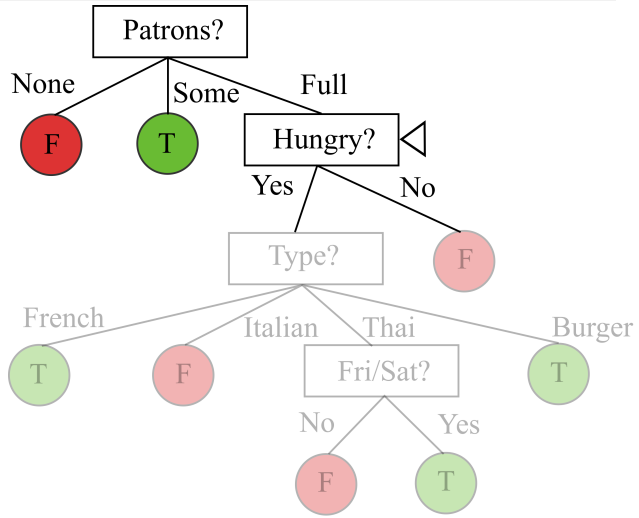
function DecisionTreeLearning (*examples*, *attributes*, *parent_examples*) **returns** tree

if *examples* is empty **then return** `Plurality-Value`(*parent_examples*)
else if all *examples* have the same classification **then return** the classification
else if *attributes* is empty **then return** `Plurality-Value`(*examples*)
else
  $A \leftarrow \arg\max_{a \in attributes}$ `Importance`$(a, examples)$
  tree $\leftarrow$ a new decision tree with root test A
  for each value v_k of A **do**
    $exs \leftarrow \{e | e \in examples$ **and** $e.A = v_k\}$
    subtree $\leftarrow$ `DecisionTreeLearning`$(exs, attributes \setminus A, examples)$
    add a branch to *tree* with label $(A = v_k)$ and subtree *subtree*
  return *tree*

examples:

Example	Hun	...	WillWait
X_2	T	...	F
X_4	T	...	T
X_5	F	...	F
X_9	F	...	F
X_{10}	T	...	F
X_{12}	T	...	T

attributes: [Alt, Bar, Fri, Hun, Price, Rain, Res, Type, Est]
parent_examples: $\{X_1, \ldots, X_{12}\}$ A: "Hungry"



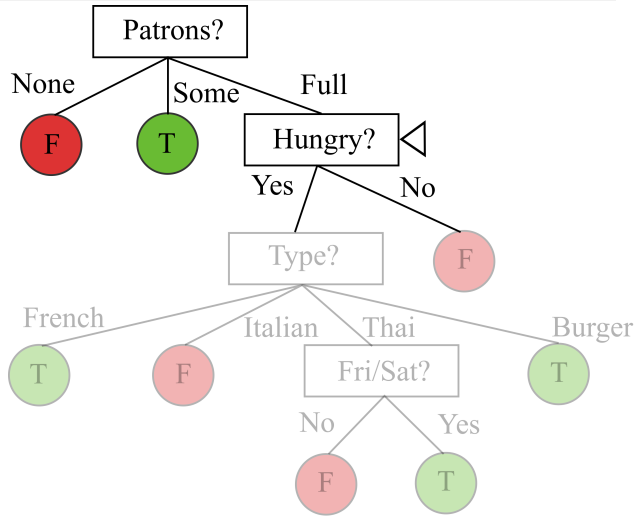
Decision Tree Learning Algorithm (DecisionTreeLearning.ipynb)

function DecisionTreeLearning (*examples*, *attributes*, *parent_examples*) **returns** tree

if *examples* is empty **then return** `Plurality-Value`(*parent_examples*)
else if all *examples* have the same classification **then return** the classification
else if *attributes* is empty **then return** `Plurality-Value`(*examples*)
else
  $A \leftarrow \arg\max_{a \in attributes}$ `Importance`$(a, examples)$
  tree $\leftarrow$ a new decision tree with root test A
  for each value v_k of A **do**
    $exs \leftarrow \{e | e \in examples \text{ and } e.A = v_k\}$
    subtree $\leftarrow$ `DecisionTreeLearning`$(exs, attributes \setminus A, examples)$
    add a branch to *tree* with label $(A = v_k)$ and subtree *subtree*
  return *tree*

examples: $\{X_2, X_4, X_5, X_9, X_{10}, X_{12}\}$
attributes: [Alt, Bar, Fri, Hun, Price, Rain, Res, Type, Est]
parent_examples: $\{X_1, \ldots, X_{12}\}$
A: "Hungry" v_k: "No"

	Example	Hun	...	WillWait
exs:	X_5	F	...	F
	X_9	F	...	F



Decision Tree Learning Algorithm (DecisionTreeLearning.ipynb)

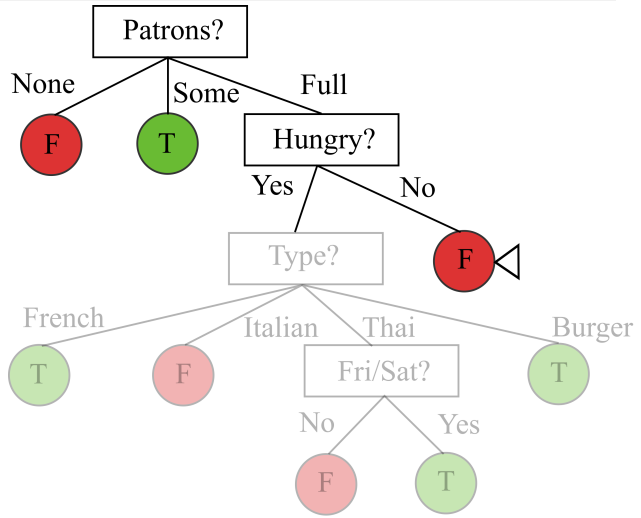
function DecisionTreeLearning (*examples*, *attributes*, *parent_examples*) **returns** tree

if *examples* is empty **then return** `Plurality-Value`(*parent_examples*)
else if all *examples* have the same classification **then return** the classification
else if *attributes* is empty **then return** `Plurality-Value`(*examples*)
else
  $A \leftarrow \arg\max_{a \in attributes}$ `Importance`$(a, examples)$
  tree $\leftarrow$ a new decision tree with root test A
  for each value v_k of A **do**
    $exs \leftarrow \{e | e \in examples$ **and** $e.A = v_k\}$
    subtree $\leftarrow$ `DecisionTreeLearning`$(exs, attributes \setminus A, examples)$
    add a branch to *tree* with label $(A = v_k)$ and subtree *subtree*
  return *tree*

examples:

Example	Hun	...	WillWait
X_5	F	...	F
X_9	F	...	F

attributes: [Alt, Bar, Fri, Price, Rain, Res, Type, Est]
parent_examples: $\{X_2, X_4, X_5, X_9, X_{10}, X_{12}\}$



Decision Tree Learning Algorithm (DecisionTreeLearning.ipynb)

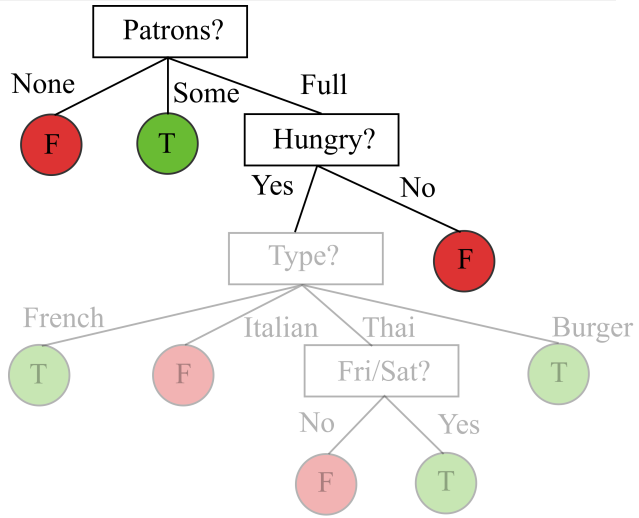
function DecisionTreeLearning (*examples*, *attributes*, *parent_examples*) **returns** tree

if *examples* is empty **then return** `Plurality-Value`(*parent_examples*)
else if all *examples* have the same classification **then return** the classification
else if *attributes* is empty **then return** `Plurality-Value`(*examples*)
else
  $A \leftarrow \arg\max_{a \in attributes}$ `Importance`$(a, examples)$
  tree $\leftarrow$ a new decision tree with root test A
  for each value v_k of A **do**
    exs $\leftarrow \{e | e \in examples$ **and** $e.A = v_k\}$
    subtree $\leftarrow$ `DecisionTreeLearning`(*exs*, *attributes* $\setminus$ A, *examples*)
    add a branch to *tree* with label $(A = v_k)$ and subtree *subtree*
  return *tree*

examples: $\{X_2, X_4, X_5, X_9, X_{10}, X_{12}\}$
attributes: [Alt, Bar, Fri, Hun, Price, Rain, Res, Type, Est]
parent_examples: $\{X_1, \ldots, X_{12}\}$
A: "Hungry" v_k: "Yes"

exs:

Example	Hun	...	WillWait
X_2	T	...	F
X_4	T	...	T
X_{10}	T	...	F
X_{12}	T	...	T

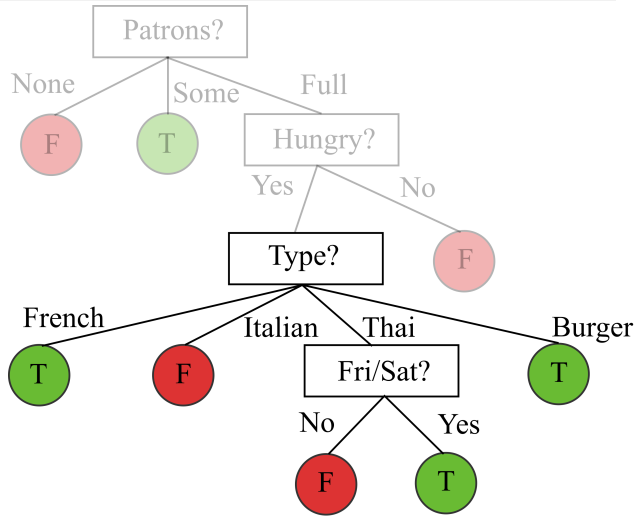


Decision Tree Learning Algorithm (DecisionTreeLearning.ipynb)

function DecisionTreeLearning (*examples*, *attributes*, *parent_examples*) **returns** tree

if *examples* is empty **then return** `Plurality-Value`(*parent_examples*)
else if all *examples* have the same classification **then return** the classification
else if *attributes* is empty **then return** `Plurality-Value`(*examples*)
else
  $A \leftarrow \arg\max_{a \in attributes}$ `Importance`$(a, examples)$
  tree $\leftarrow$ a new decision tree with root test A
  for each value v_k of A **do**
    $exs \leftarrow \{e | e \in examples$ **and** $e.A = v_k\}$
    subtree $\leftarrow$ `DecisionTreeLearning`$(exs, attributes \setminus A, examples)$
    add a branch to *tree* with label $(A = v_k)$ and subtree *subtree*
  return *tree*

(steps omitted)
examples: $\{X_2, X_4, X_{10}, X_{12}\}$
attributes: [Alt, Bar, Fri, Price, Rain, Res, Type, Est]
parent_examples: $\{X_2, X_4, X_5, X_9, X_{10}, X_{12}\}$

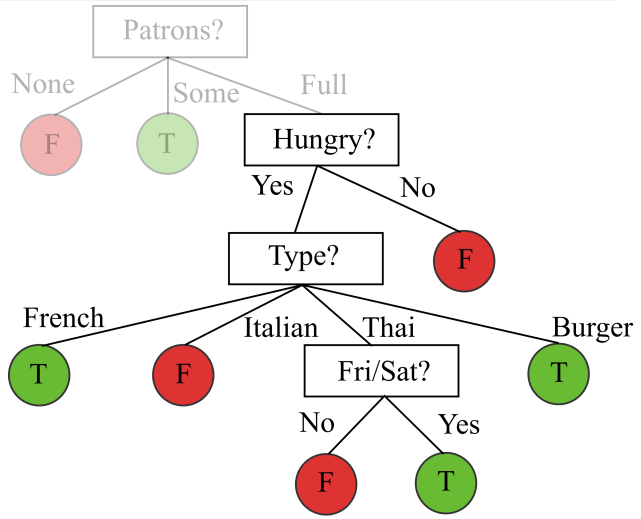


Decision Tree Learning Algorithm (DecisionTreeLearning.ipynb)

function DecisionTreeLearning (*examples*, *attributes*, *parent_examples*) **returns** tree

if *examples* is empty **then return** `Plurality-Value`(*parent_examples*)
else if all *examples* have the same classification **then return** the classification
else if *attributes* is empty **then return** `Plurality-Value`(*examples*)
else
  $A \leftarrow \arg\max_{a \in attributes}$ `Importance`$(a, examples)$
  tree $\leftarrow$ a new decision tree with root test A
  for each value v_k of A **do**
    $exs \leftarrow \{e | e \in examples$ **and** $e.A = v_k\}$
    subtree $\leftarrow$ `DecisionTreeLearning`$(exs, attributes \setminus A, examples)$
    add a branch to *tree* with label $(A = v_k)$ and subtree *subtree*
  return *tree*

examples: $\{X_2, X_4, X_5, X_9, X_{10}, X_{12}\}$
attributes: [Alt, Bar, Fri, Hun, Price, Rain, Res, Type, Est]
parent_examples: $\{X_1, \ldots, X_{12}\}$
A: "Hungry"

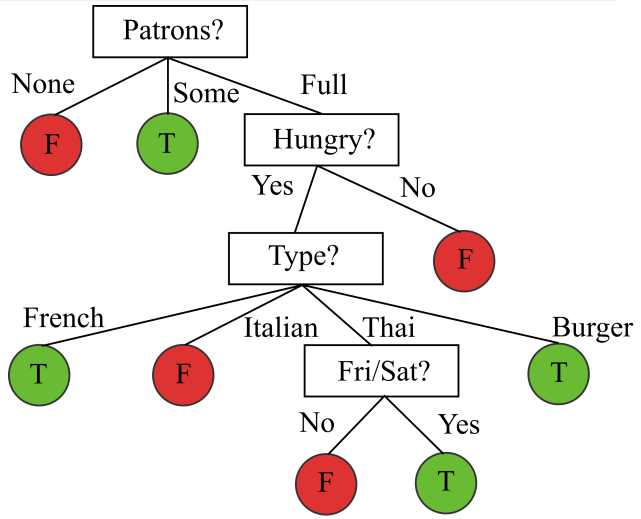


Decision Tree Learning Algorithm (DecisionTreeLearning.ipynb)

function DecisionTreeLearning (*examples*, *attributes*, *parent_examples*) **returns** tree

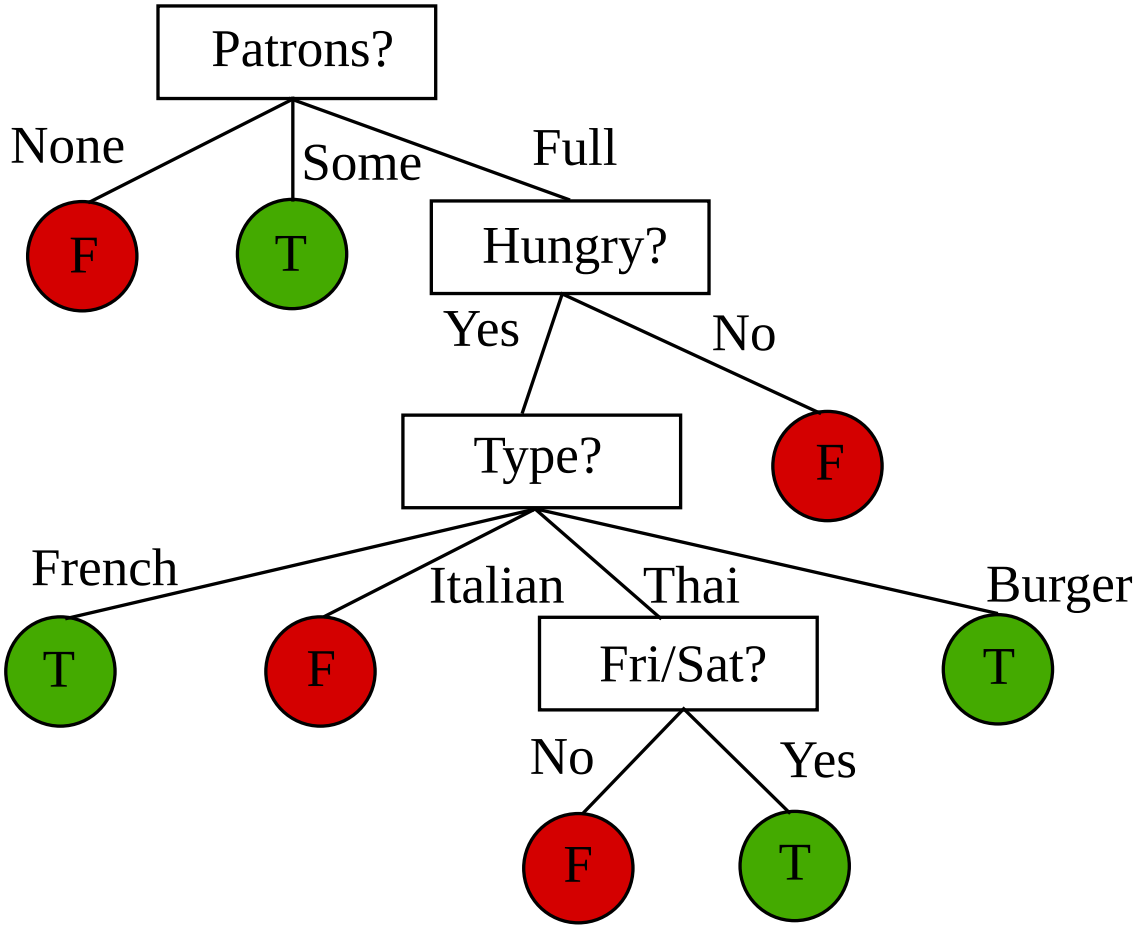
if *examples* is empty **then return** `Plurality-Value`(*parent_examples*)
else if all *examples* have the same classification **then return** the classification
else if *attributes* is empty **then return** `Plurality-Value`(*examples*)
else
  $A \leftarrow \arg\max_{a \in attributes}$ `Importance`$(a, examples)$
  tree $\leftarrow$ a new decision tree with root test A
  for each value v_k of A **do**
    $exs \leftarrow \{e | e \in examples$ **and** $e.A = v_k\}$
    subtree $\leftarrow$ `DecisionTreeLearning`$(exs, attributes \setminus A, examples)$
    add a branch to *tree* with label $(A = v_k)$ and subtree *subtree*
  return *tree*

examples: $\{X_1, \ldots, X_{12}\}$
attributes: [Alt, Bar, Fri, Hun, Pat, Price, Rain, Res, Type, Est]
parent_examples: $\{X_1, \ldots, X_{12}\}$
A: "Patrons"



Result from the Training Set (DecisionTreeLearning.ipynb)

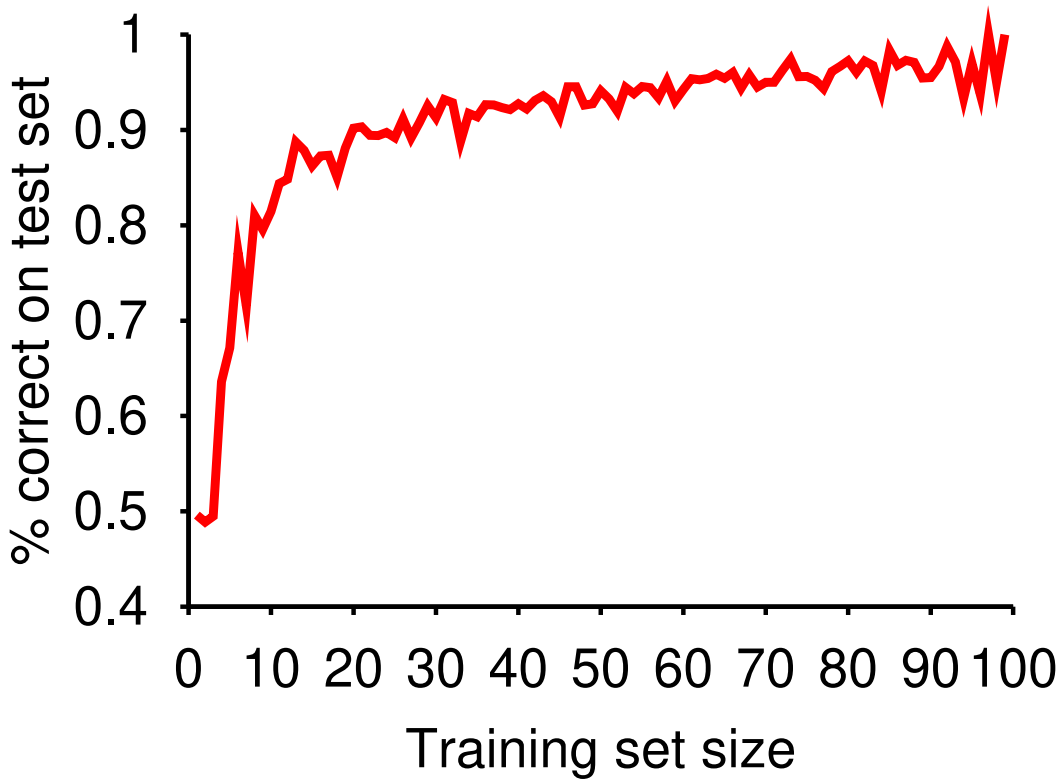
Decision tree learned from the 12 examples:



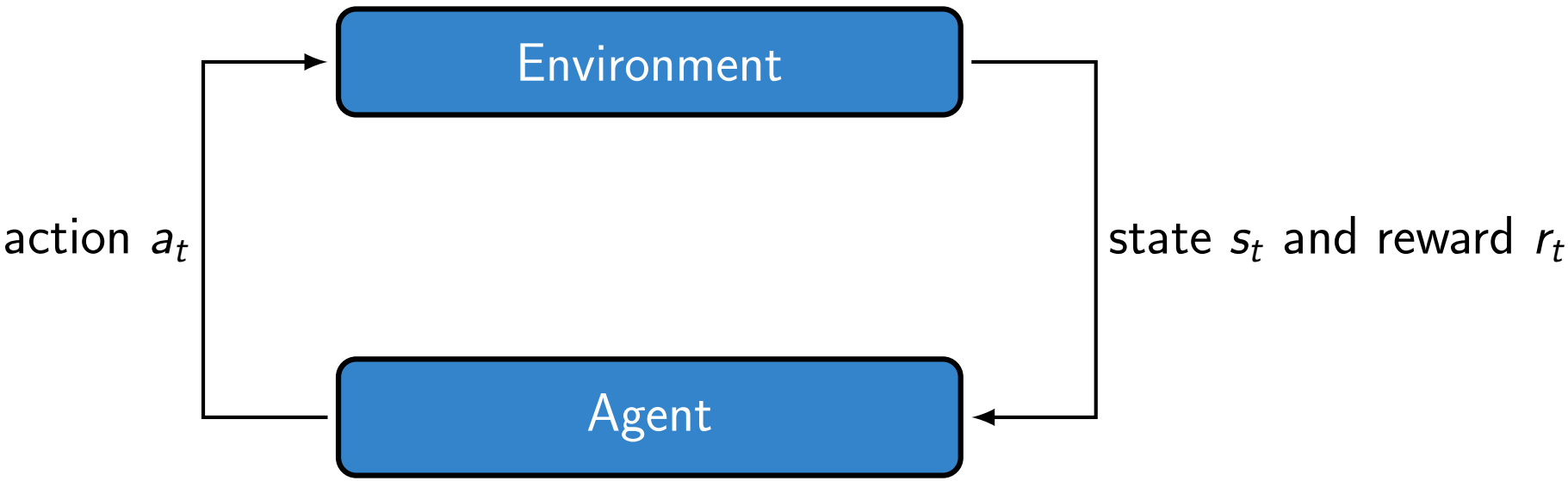
Substantially simpler than "true" tree – a more complex hypothesis is not justified by small amount of data.

Learning Curve

- We can evaluate the accuracy of a learning algorithm with a **learning curve**.
- We have 100 examples, which we split into a training set and a test set.
- We start with a training set of size 1 and increase it up to size 99.
- For each training/test ratio, 20 random splits are used and averaged.



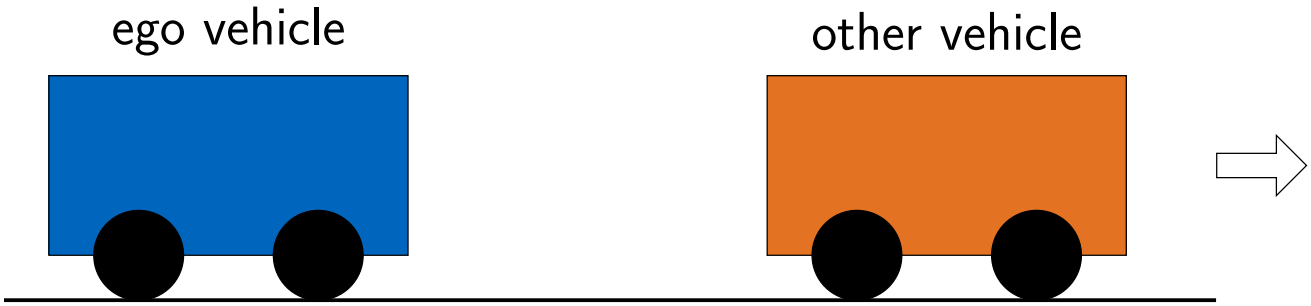
What is Reinforcement Learning (RL)?



Example: Vehicle Following Task

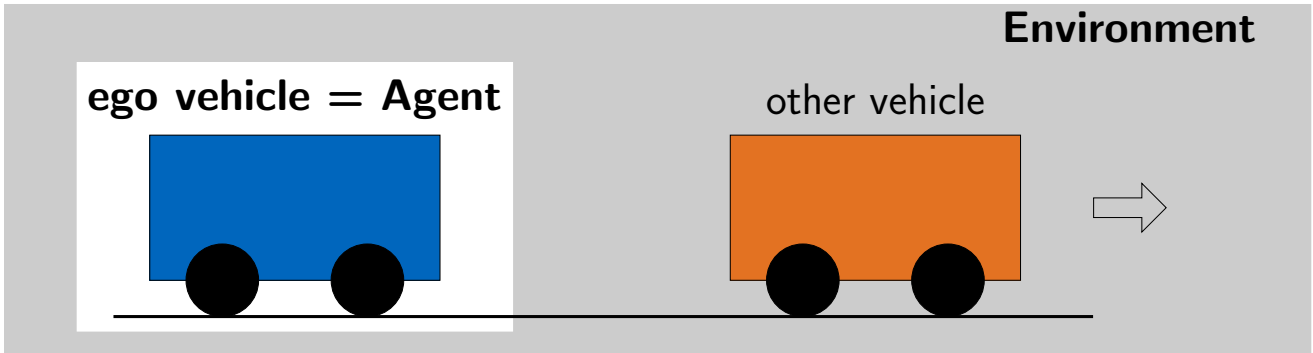
Task

The agent should follow the other vehicle while keeping a safe distance.



Agent

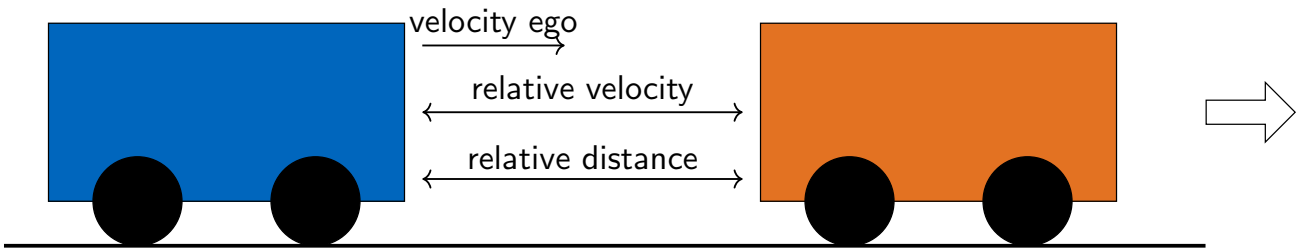
The agent is the considered entity perceiving its environment and acting upon it.



Example: State and Action Selection

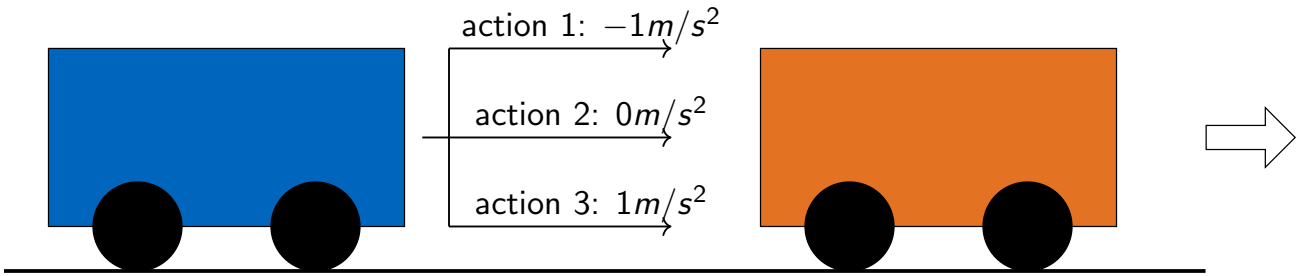
State

The agent observes its state from the environment.



Action

The agent acts on the environment through possible actions.

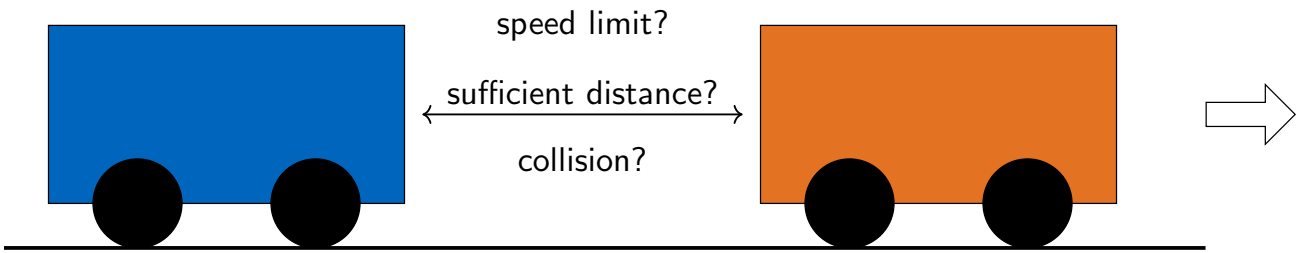


Example: Reward Design

Reward

The reward indicates how beneficial the last action was with respect to solving the task.

Tweedback: What could be components of the reward for this example?



Policy

In contrast to finding optimal decisions using deterministic policies for exploitation (see previous lecture), reinforcement learning requires stochastic policies for exploration:

Deterministic and stochastic policy

- Deterministic policy $\pi : S \to A$ (the state determines the action);
- Stochastic policy $\pi : S \times A \to [0, 1]$ under the constraint $\sum_{a \in A} \pi(a, s) = 1$ (for each action the policy returns a probability).

Exploration and exploitation

- *Exploration*: Randomly choosing the next action to create new experience for improving the policy.
- *Exploitation*: Using the existing policy to decide for the next action.

Trade-off between exploration and exploitation necessary to find an optimal policy.

Objective and Utility

The objective is to learn a policy π that maximizes the expected rewards (aka expected utility $U_\pi(s)$) for each state s:

$$\max_{\pi \in \Pi} U_\pi(s),$$

where Π is the policy space. The utility (aka value function) can be derived analogously to MDPs, except that a probabilistic policy is learned:

Value function

Probabilistic policy:

$$U_\pi(s) = \sum_{a \in A(s)} \pi(a, s) \Big(\sum_{s'} P(s'|s, a)[R(s, a, s') + \gamma U_\pi(s')] \Big).$$

Optimal policy:

$$U^*(s) = \max_{a \in A(s)} \sum_{s'} P(s'|s, a)[R(s, a, s') + \gamma U^*(s')].$$

Q-Function

The Q-function can also be derived analogously to MDPs, except that a probabilistic policy is learned:

Q-function

Probabilistic policy:

$$Q_\pi(s,a) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma \sum_{a' \in A(s')} \pi(s',a') Q_\pi(s',a')].$$

Optimal policy:

$$Q^*(s,a) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma \max_{a' \in A(s')} Q^*(s',a')].$$

In contrast to the utility, no transition model is necessary for obtaining the optimal policy:

$$\pi^*(s) = \arg\max_a Q^*_\pi(s,a)$$

Problems of the Q-Value Update

Observation: When using the Bellmann equation of the Q-function

$$Q_i(s,a) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma \sum_{a' \in A(s')} \pi(s',a') Q_{i-1}(s',a')],$$

we only consider the most recent information and ignore prior learned knowledge, since $Q_{i-1}(s,a)$ is completely replaced by $Q_i(s,a)$.

Consequence: Learning becomes very noisy.

Solution: Update with a learning rate $\Rightarrow$ Q-Learning.

Q-Learning

Core idea: Learning the Q-function by updating it incrementally.

$$Q_{i+1}(s,a) = Q_i(s,a) + \alpha \cdot \Big[\underbrace{\sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma \max_{a' \in A(s')} Q_i(s',a'))}_{①} - \underbrace{Q_i(s,a)}_{②} \Big]$$

α: learning rate

①: recent value

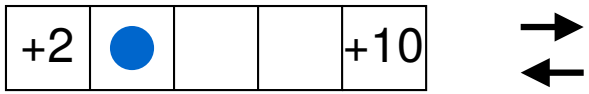
②: prior value

① − ② is called the temporal difference (TD) error.

Simple Case: Q-Learning for Deterministic Problems (1)

$$Q_{i+1}(s,a) = Q_i(s,a) + \underbrace{\alpha}_{\text{learning rate}} \underbrace{[R(s,a,s') + \gamma \max_{a' \in A(s')} Q_i(s',a') - Q_i(s,a)]}_{\text{TD Error}}$$

Example:



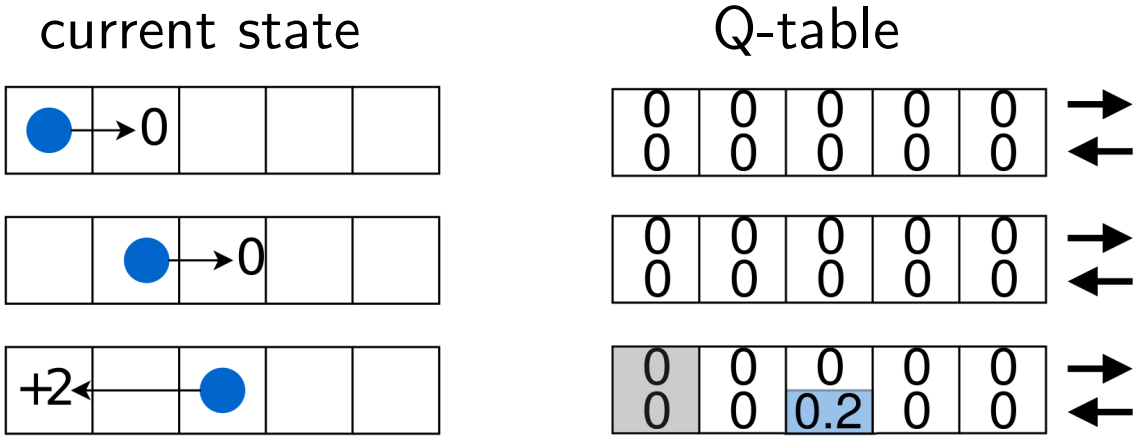
- An agent can move left or right in a 5x1 grid world.
  - Right: one step to the right.
  - Left: takes agent to leftmost cell.
- Rewards unknown to the agent are shown in the figure.
- The agent samples its policy randomly to explore the unknown grid world. We choose $\alpha = 0.1$, $\gamma = 0.95$.
- **Question**: Why is exploration important in the learning phase?

Simple Case: Q-Learning for Deterministic Problems (2)

Special Case:

$$Q_{i+1}(s,a) = Q_i(s,a) + \underbrace{\alpha}_{\text{learning rate}} \underbrace{[R(s,a,s') + \gamma \max_{a' \in A(s')} Q_i(s',a') - Q_i(s,a)]}_{\text{TD Error}}$$

Example:



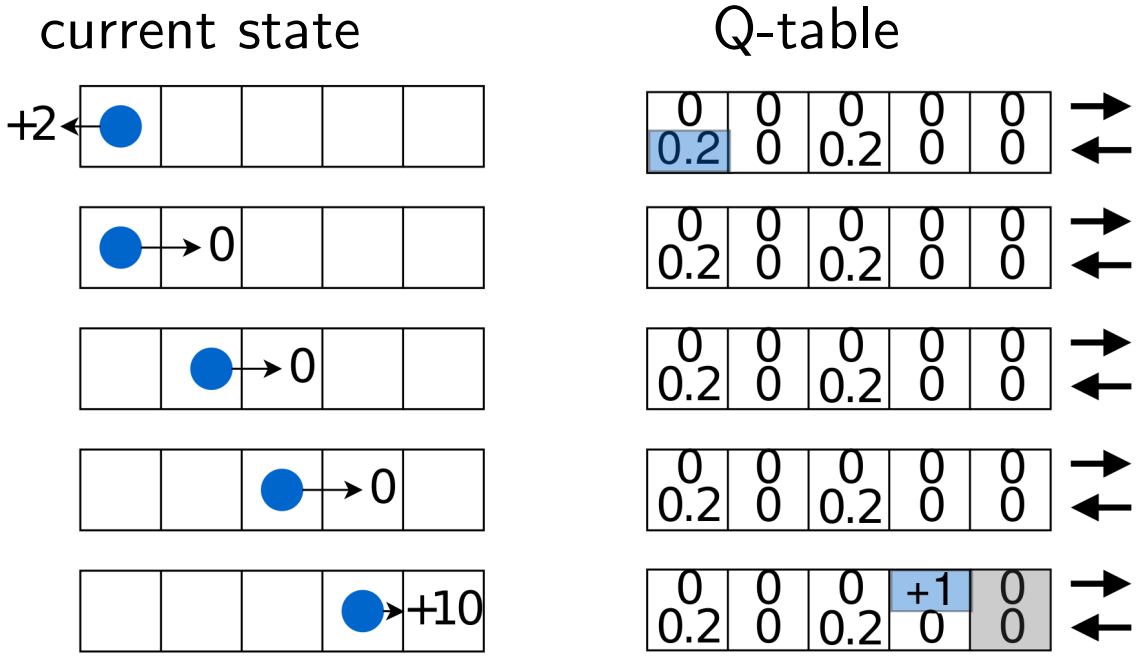
$$0 + 0.1 \cdot [2 + 0.95 \cdot \max(\,0\,,\,0\,) - 0] = +0.2$$

Simple Case: Q-Learning for Deterministic Problems (3)

Special Case:

$$Q_{i+1}(s,a) = Q_i(s,a) + \underbrace{\alpha}_{\text{learning rate}} \underbrace{[R(s,a,s') + \gamma \max_{a' \in A(s')} Q_i(s',a') - Q_i(s,a)]}_{\text{TD Error}}$$

Example:



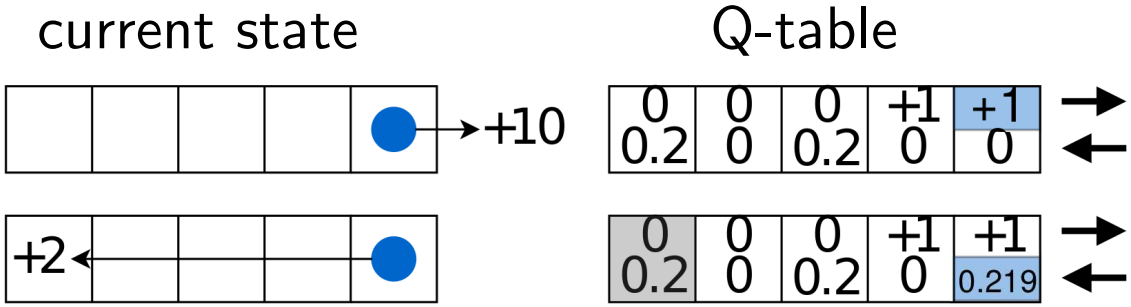
$$0 + 0.1 \cdot [10 + 0.95 \cdot \max(\ 0\ ,\ 0\ ) - 0] = +1.0$$

Simple Case: Q-Learning for Deterministic Problems (4)

Special Case:

$$Q_{i+1}(s,a) = Q_i(s,a) + \underbrace{\alpha}_{\text{learning rate}} \underbrace{[R(s,a,s') + \gamma \max_{a' \in A(s')} Q_i(s',a') - Q_i(s,a)]}_{\text{TD Error}}$$

Example:



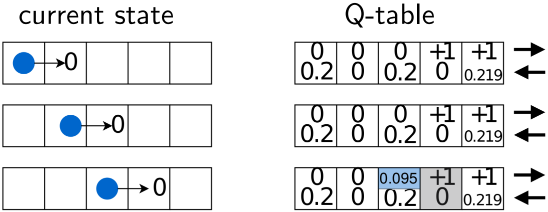
$$0 + 0.1 \cdot [2 + 0.95 \cdot \max(\ 0\ ,\ 0.2\ ) - 0] = +0.219$$

Simple Case: Q-Learning for Deterministic Problems (5)

Special Case:

$$Q_{i+1}(s,a) = Q_i(s,a) + \underbrace{\alpha}_{\uparrow \text{ learning rate}} \Big[\underbrace{R(s,a,s') + \gamma \max_{a' \in A(s')} Q_i(s',a') - Q_i(s,a)}_{\text{TD Error}}\Big]$$

Example:



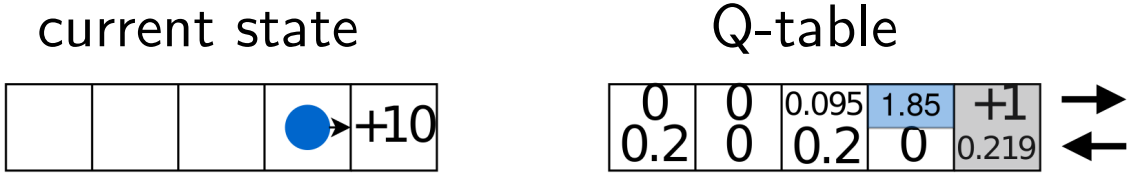
$$0 + 0.1 \cdot [0 + 0.95 \cdot \max(\,0\,,\,1.0\,) - 0] = +0.095$$

Simple Case: Q-Learning for Deterministic Problems (6)

Special Case:

$$Q_{i+1}(s,a) = Q_i(s,a) + \underbrace{\alpha}_{\text{learning rate}} \underbrace{[R(s,a,s') + \gamma \max_{a' \in A(s')} Q_i(s',a') - Q_i(s,a)]}_{\text{TD Error}}$$

Example:



$$1.0 + 0.1 \cdot [10 + 0.95 \cdot \max(\ 1.0\ ,\ 0.219\ ) - 1.0] = +1.85$$

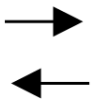
Simple Case: Q-Learning for Deterministic Problems (7)

Special Case:

$$Q_{i+1}(s,a) = Q_i(s,a) + \underbrace{\alpha}_{\text{learning rate}} \underbrace{\left[R(s,a,s') + \gamma \max_{a' \in A(s')} Q_i(s',a') - Q_i(s,a)\right]}_{\text{TD Error}}$$

Example:

62.73	66.31	70.82	82.63	84.51
60.12	61.22	63.43	64.75	66.49



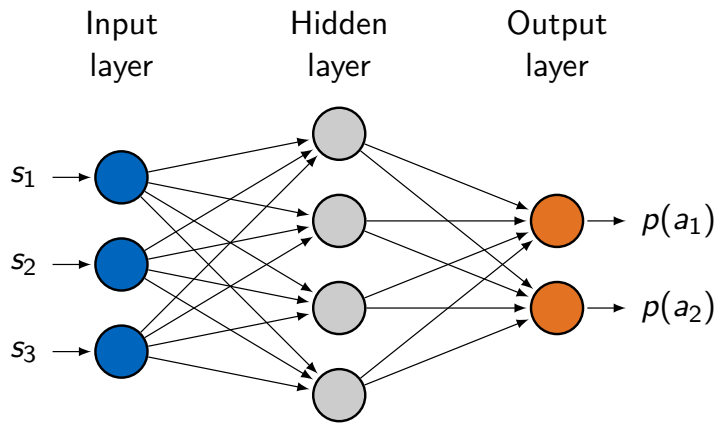
After many steps, we obtain the Q-table shown above.

RL versus Deep RL

RL

	a_1	a_2
s_1		
s_2		
s_3		

Deep RL

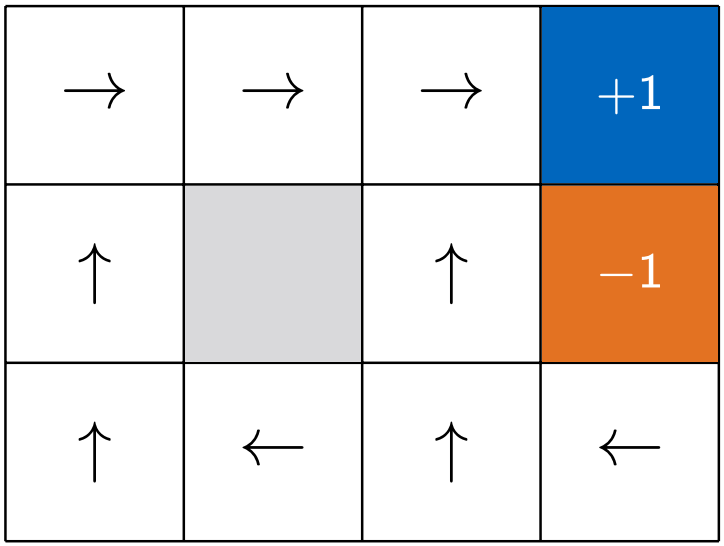


Representation of policy or Q-function

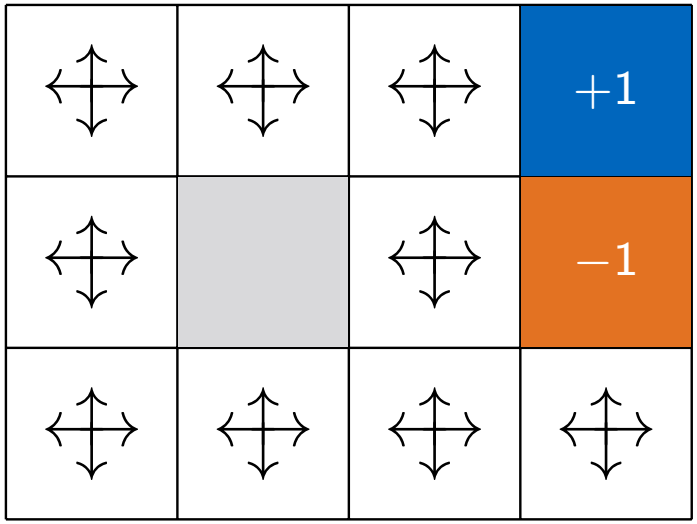
- *RL*: Look-up table
- *Deep RL*: Neural network (NN) of arbitrary architecture (e.g., convolutional NN, recurrent NN, ...)

On-policy versus off-policy

Target policy



Behavioral policy



RL algorithm properties

- *on-policy*: Learning update uses only data which was collected with the recent policy (i.e. target policy equals behavioral policy).
- *off-policy*: Learning update can use any training data (i.e. target policy differs from behavioral policy).

Definitions for RL Algorithms

Definition of RL algorithm classes

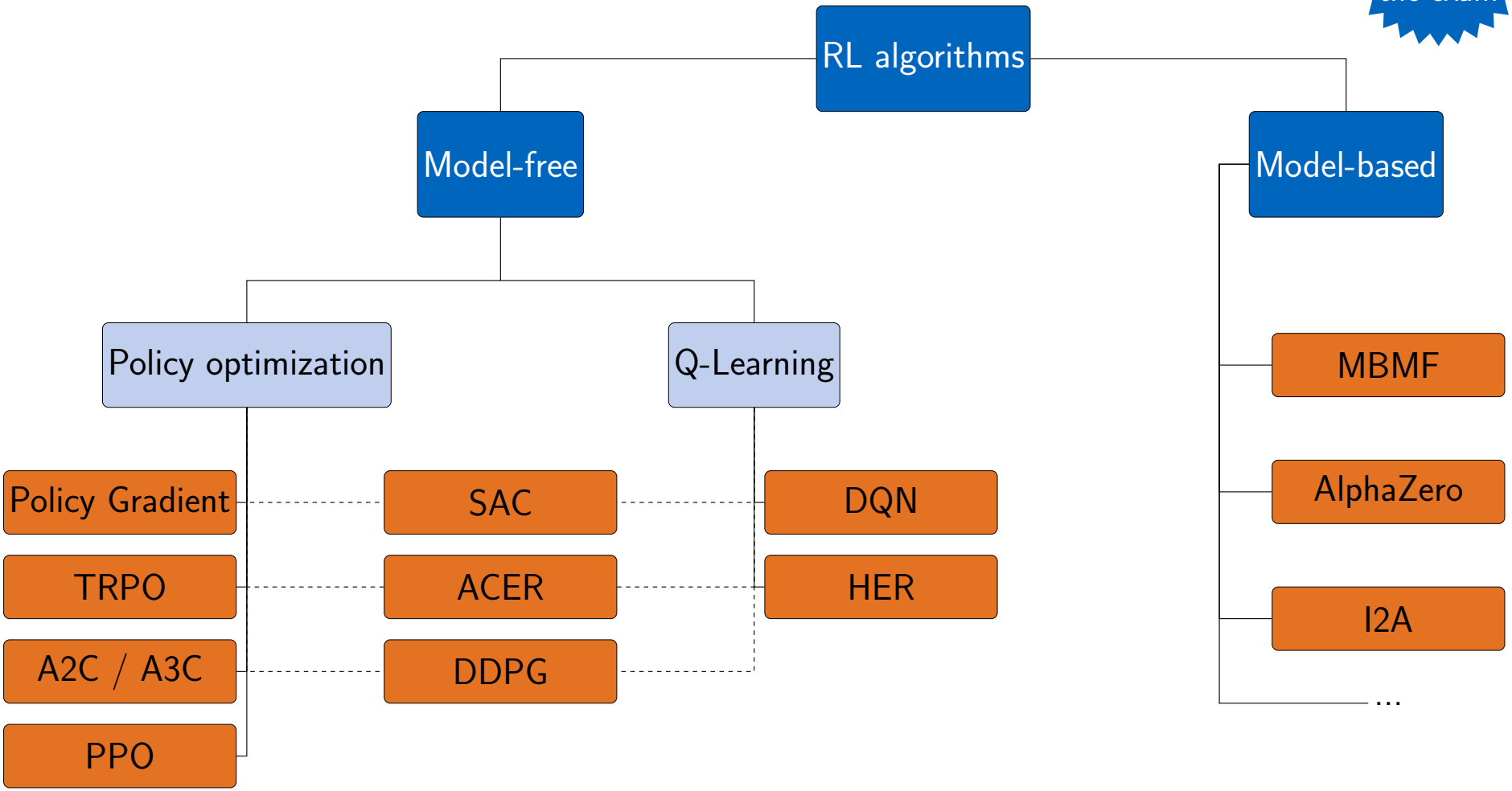
- *model-based*: State transition model for the environment is used (i.e., $P(s'|s,a)$ is known or empirically estimated).
- *model-free*: Learning from sampling and simulation with a "black-box" environment without explicitly considering transition probabilities.
  - *policy optimization*: Explicit representation and learning of the policy.
  - *Q-learning*: Learning of the Q-function and inferring the policy.

Examples:

- I2A (Imagination-augmented agents): model-based.
- DQN (Deep Q-network): model-free, Q-learning, off-policy.
- PPO (Proximal policy optimization): model-free, policy optimization, on-policy.

Overview of Prominent Deep RL Algorithms[1]

Not relevant for the exam



[1] Adapted from: https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html

What Could be Problematic when using RL?

- **Adversarial attacks** can confuse policy networks.
  Example: The state is an image and modified with a small random noise (invisible for humans) which changes the agent's behavior.
- **Testing is not sufficient** for ensuring specifications.
  Example: Autonomous vehicle fails to avoid a collision in a rare traffic scenario (e.g. street is white and slippery as a milk truck leaks).
- **Random initialization** leads to initially unsafe policies.
  Example: An autonomous robot is placed in a warehouse's corner. The initial policy makes moving in all directions equally likely.
- **Random exploration** leads to unsafe states or actions during learning.
  Example: Autonomous vehicles tries a right turn on a straight.

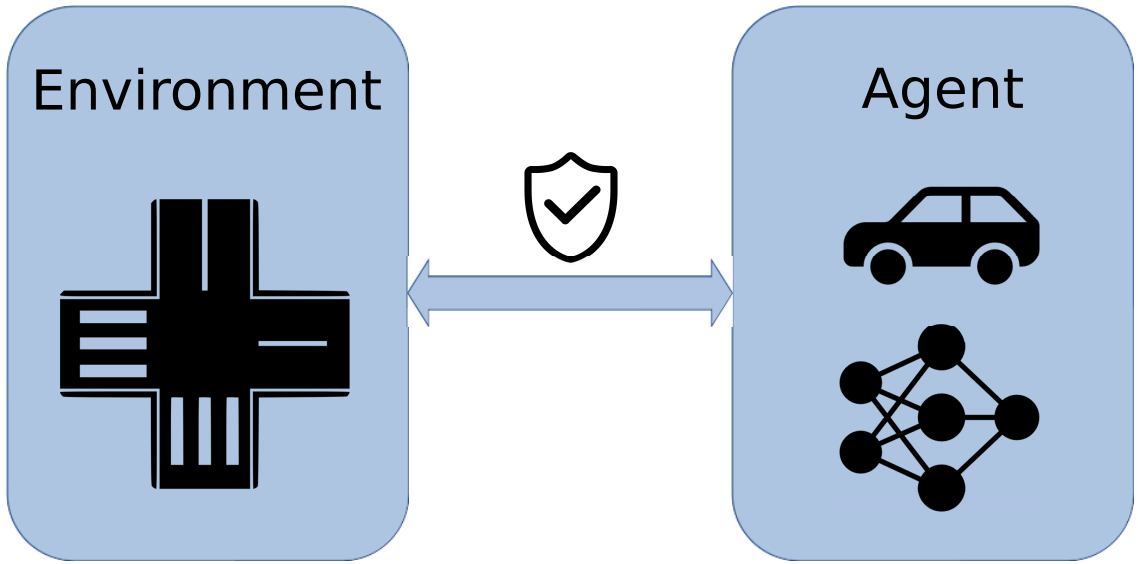
Solution (Lecture Formal Methods for Cyber-Physical Systems (IN2383))

Safe reinforcement learning where guidance with respect to safety is given to the agent during learning and/or deployment.

Case Study: Safe RL for Lane Changing[2]

Not relevant for the exam

- **Idea**: Constraining the exploration to only safe actions.
- **Advantage**: Guarantees for the possible behavior of the agent can be given without verifying the policy.



[2] H. Krasowski, X. Wang, and M. Althoff. "Safe Reinforcement Learning for Autonomous Lane Changing Using Set-Based Prediction". In: *Proc. of the 23rd IEEE International Conference on Intelligent Transportation Systems*. 2020, pp. 1–7

Task and Reward

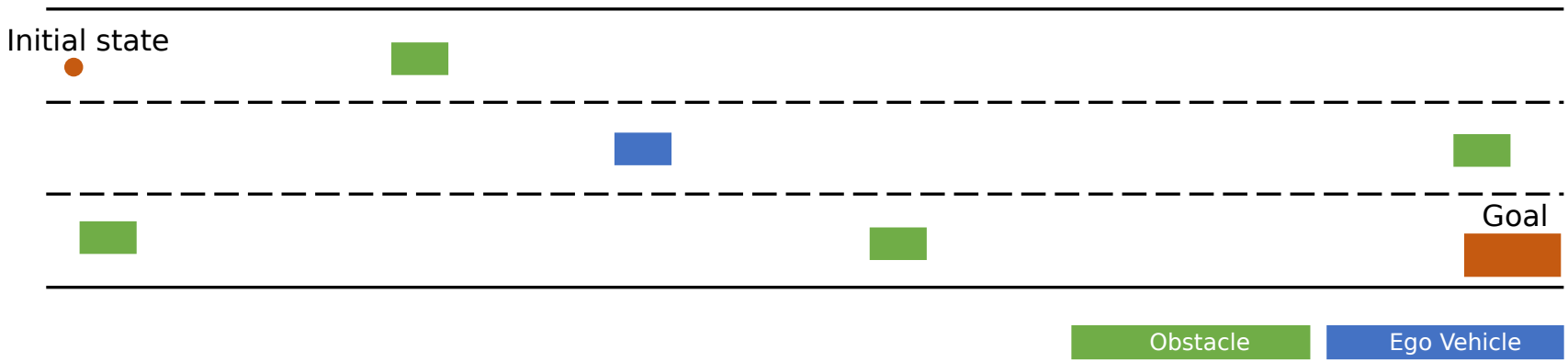
Not relevant for the exam

Task

Safely reaching a specified goal area on a highway.

Reward

$$r = r_{\text{goal\_reached}} + r_{\text{goal\_lane}} + r_{\text{closer\_to\_goal}} + r_{\text{crash}} + r_{\text{safe\_dist}}$$

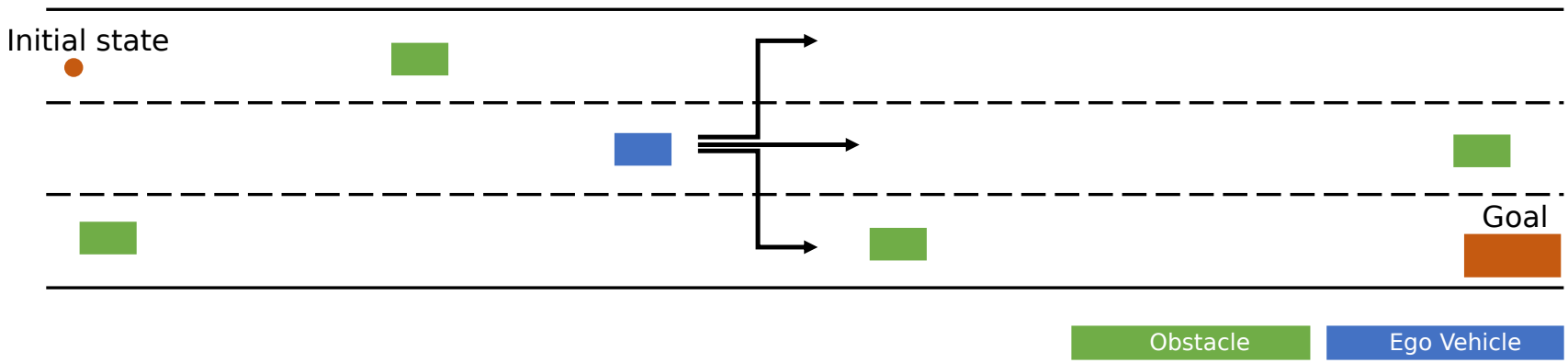


Action Space

Not relevant for the exam

Discrete actions

1. Change to right.
2. Drive ahead.
3. Change to left.
4. Execute fail-safe plan $\rightarrow$ only eligible if other three actions are unsafe.

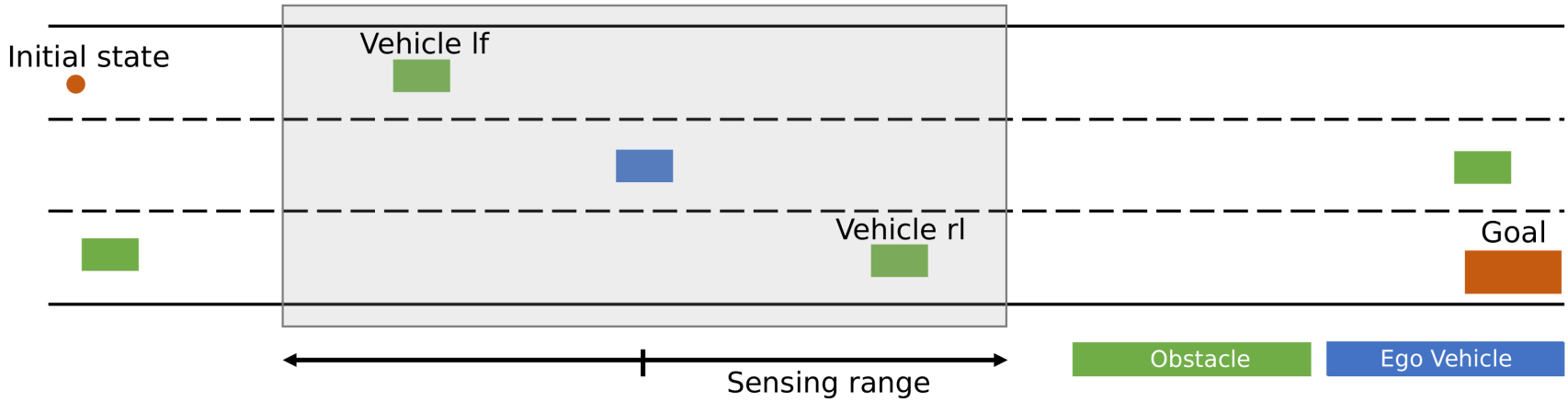


State Space

Not relevant for the exam

16 continuous variables

- 12 variables of surrounding vehicles: relative velocity and distance for each possible surrounding position.
- 2 variables for the goal: lateral and longitudinal distance.
- 2 variables for the ego vehicle: acceleration and velocity.



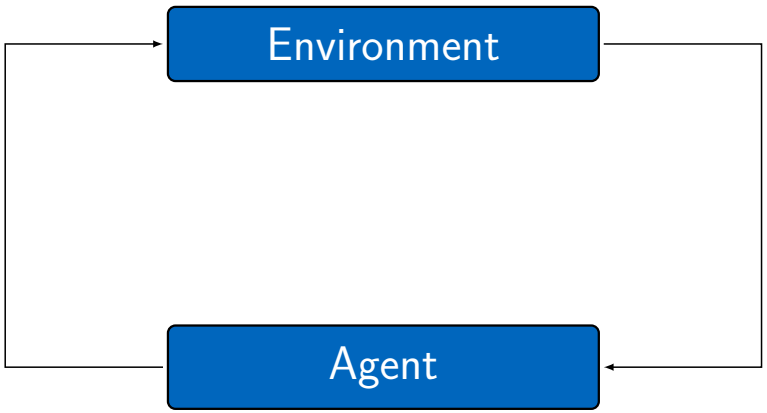
Network Types and Training Modes

Not relevant for the exam

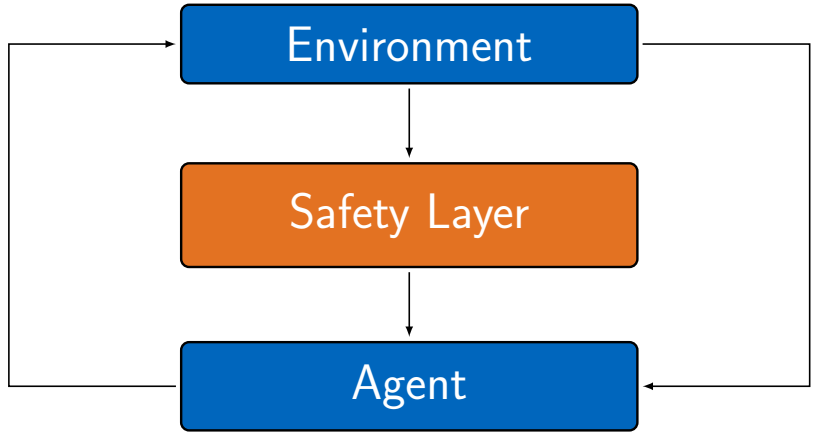
Network Types:

- Multilayer perceptron (MLP) – 3 layer with 128 neurons each
- Long short-term memory (LSTM) network – 128 neurons

Training modes:



Unsafe mode



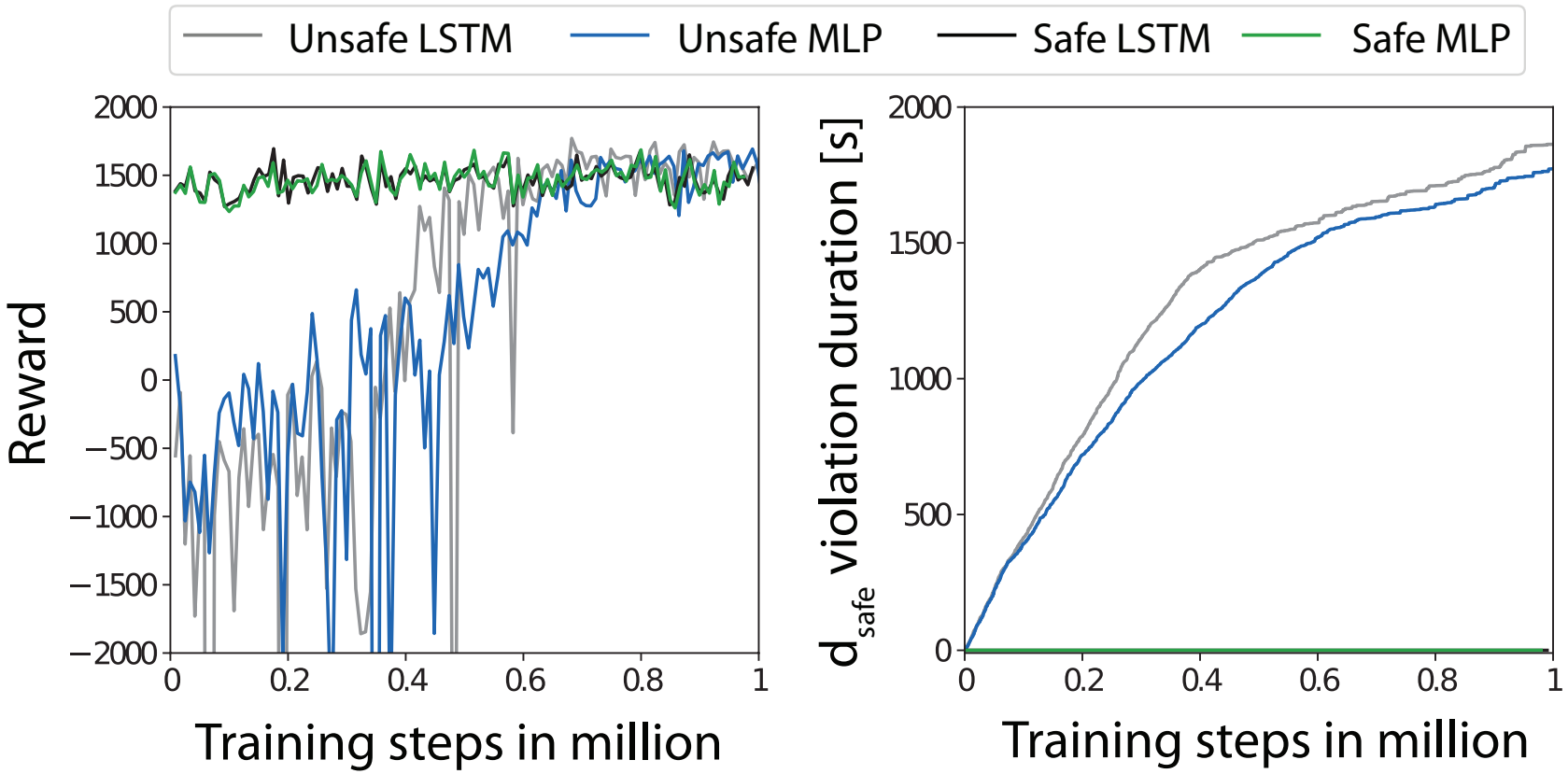
Safe mode

The safety layer will be explained in the next lecture.

Training Results

Not relevant for the exam

- No safe distance violation for safe agents.
- Safely-trained agents achieve high rewards from the beginning.
- Convergence for both network types is very similar.

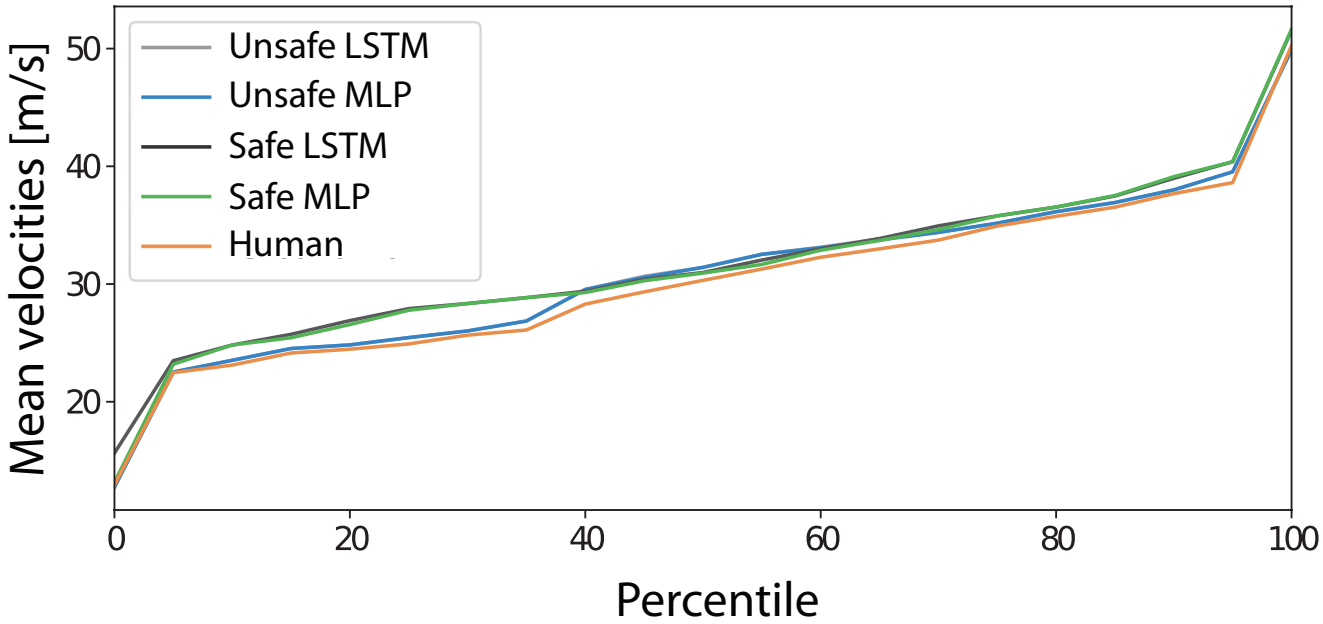


Test Results

Not relevant for the exam

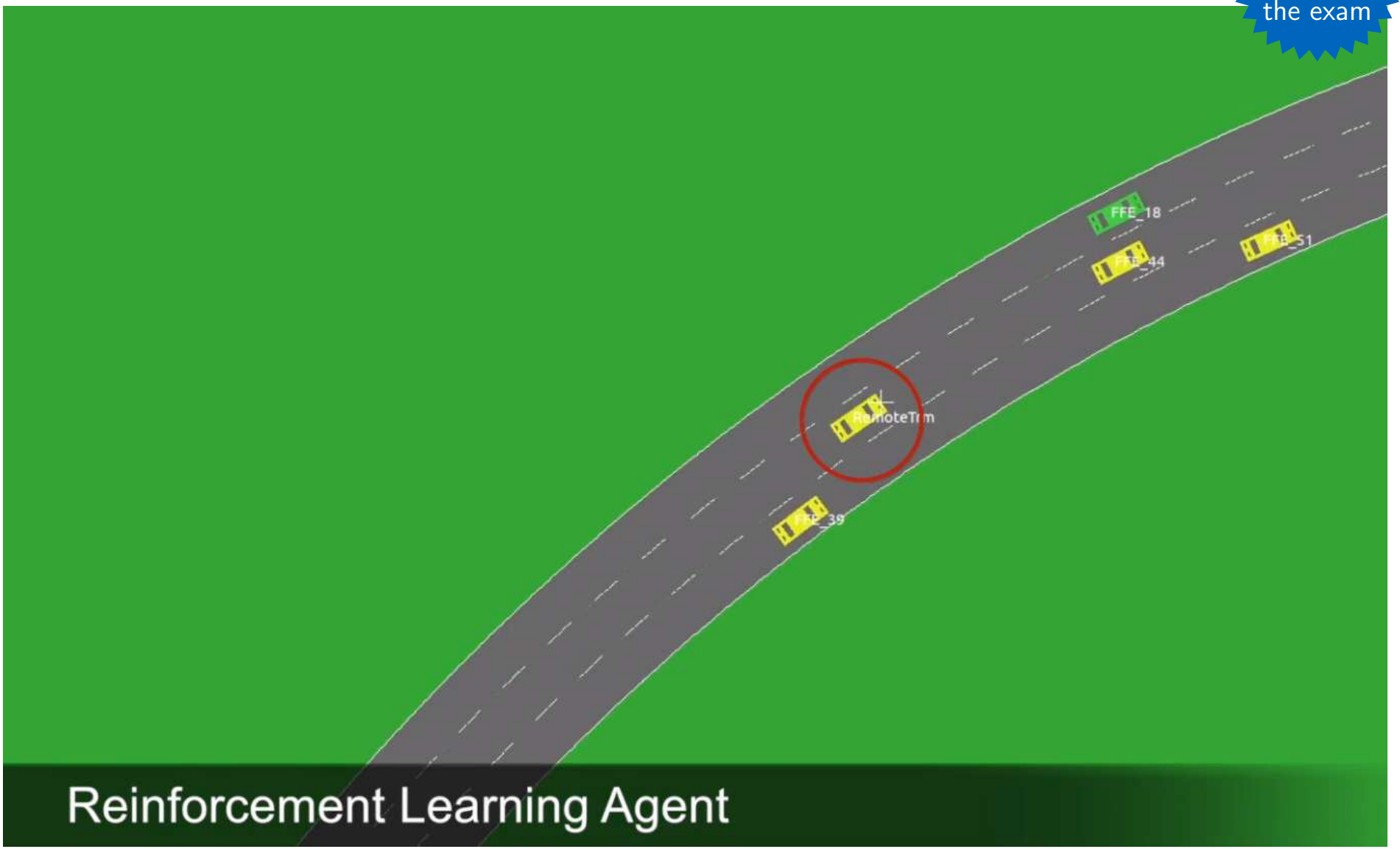
- No collisions for safely-trained agents.
- Better performance for safe LSTM agent than for safe MLP agent.

Agent	Collision	Reached goal
Unsafe LSTM	1.3%	95.0%
Unsafe MLP	0.8%	**97.1 %**
Safe LSTM	**0.0 %**	87.5%
Safe MLP	**0.0 %**	75.4%



Video

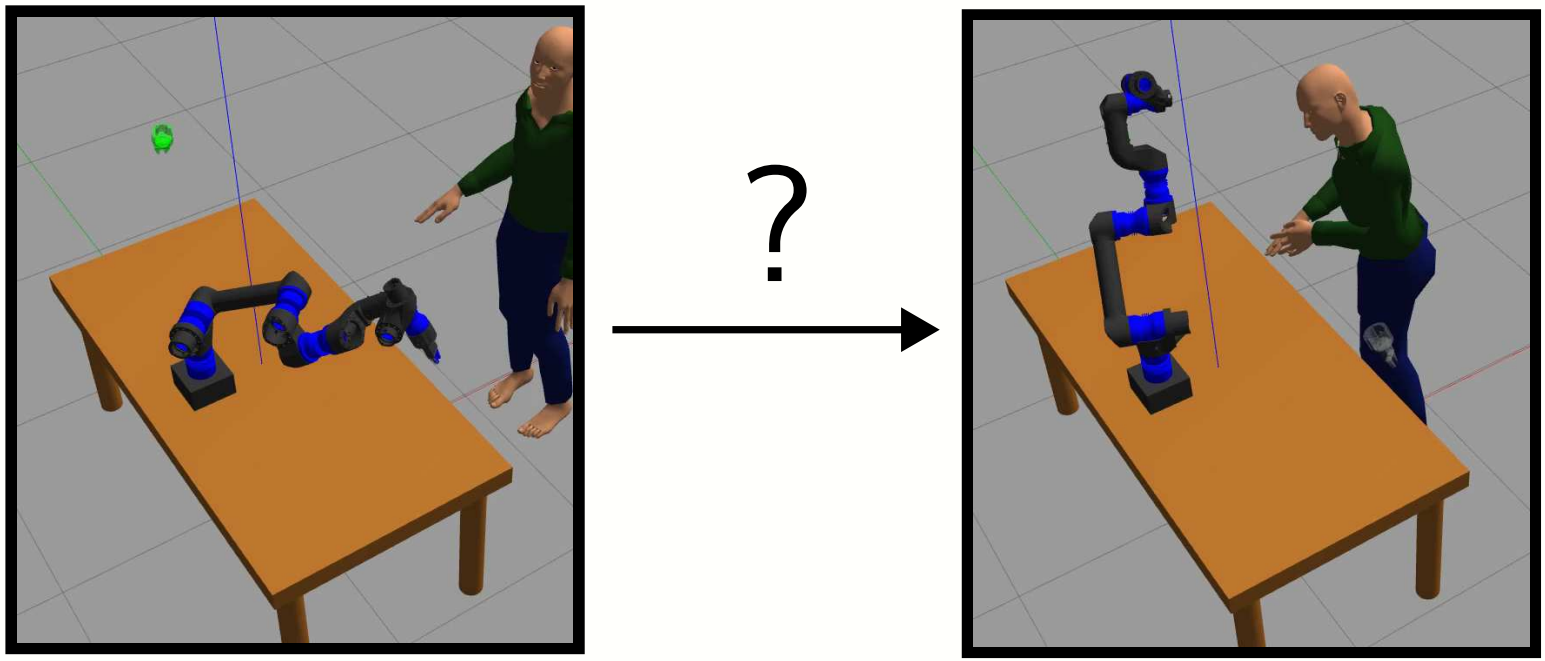
Not relevant for the exam



Safe RL for Human–Robot Collaboration

Not relevant for the exam

- **Problem statement**: How can robots work together with humans efficiently and provably safe?



Starting position

Goal position

Task and Reward

Not relevant for the exam

Task

Reach the goal and evade the human.

Reward

Dense

$$r = -\left\| s_{\text{robot}} - s_{\text{goal}} \right\|$$

- Gradient information towards the goal
- Easier for the agent to learn
- Every action is comparable in regards to reward

Sparse

$$r = \begin{cases} 0, & \text{if goal reached} \\ -1, & \text{otherwise} \end{cases}$$

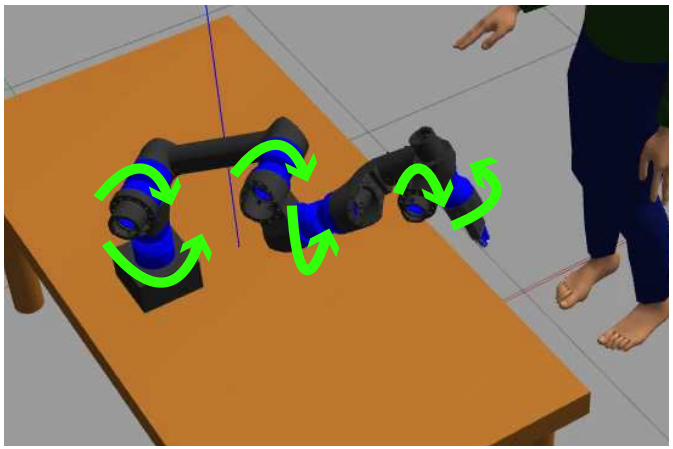
- Easy to define
- No information about the path to the goal

Action Space

Not relevant for the exam

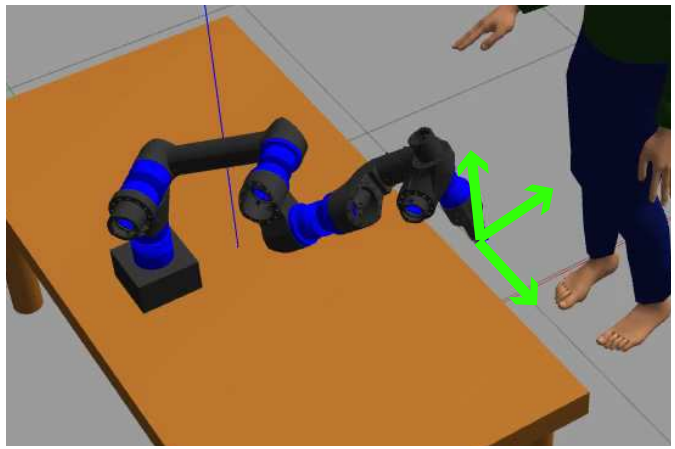
Continuous Actions

Joint Space



- n-DoF ($= 6$) actions
- Harder to learn
- More complex maneuvers possible

End-effector Space



- 3 actions
- Easier to learn
- Only EEF position controllable

State Space

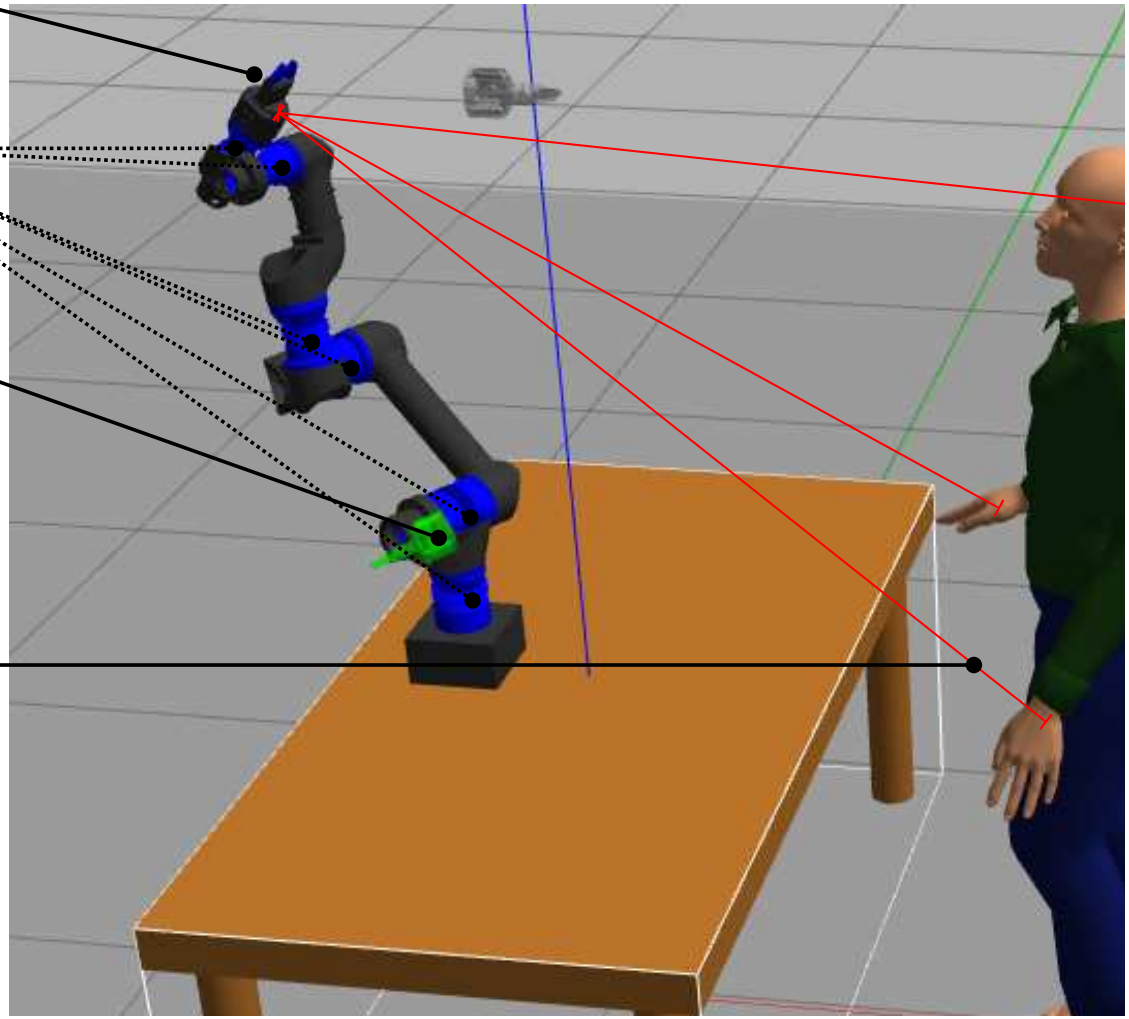
Not relevant for the exam

End effector position

Joint position and velocity

Goal joint position

Distance from end effector to wrist and head position



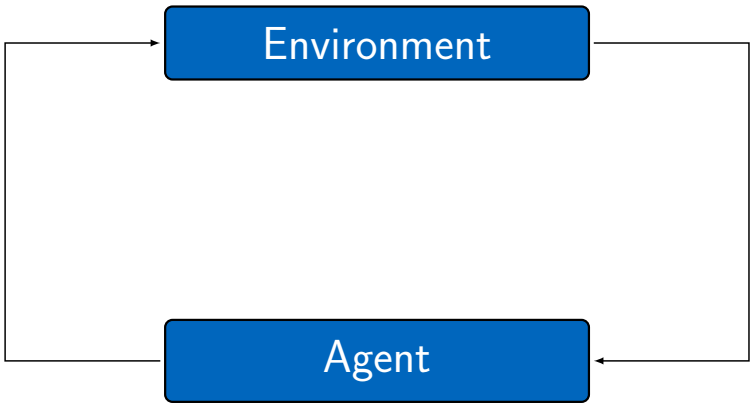
Algorithm Design

Not relevant for the exam

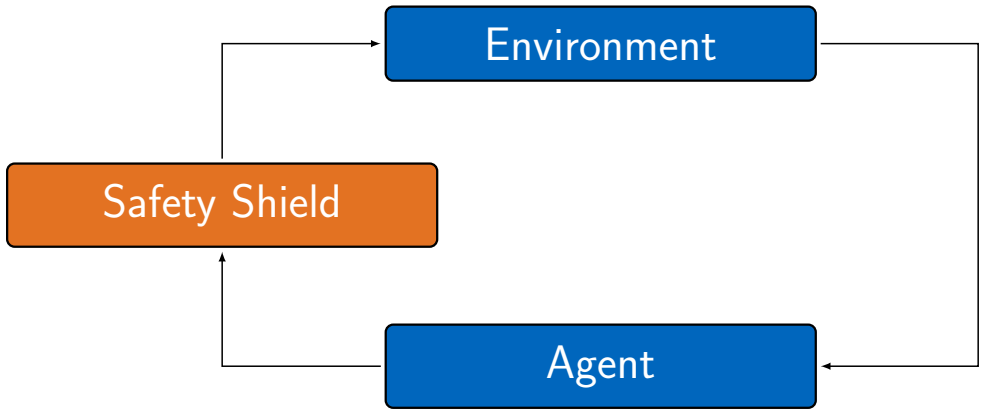
RL algorithm:

- Soft Actor-Critic (SAC) – off-policy model-free RL
- Sparse reward
- Joint space actions

Training modes:



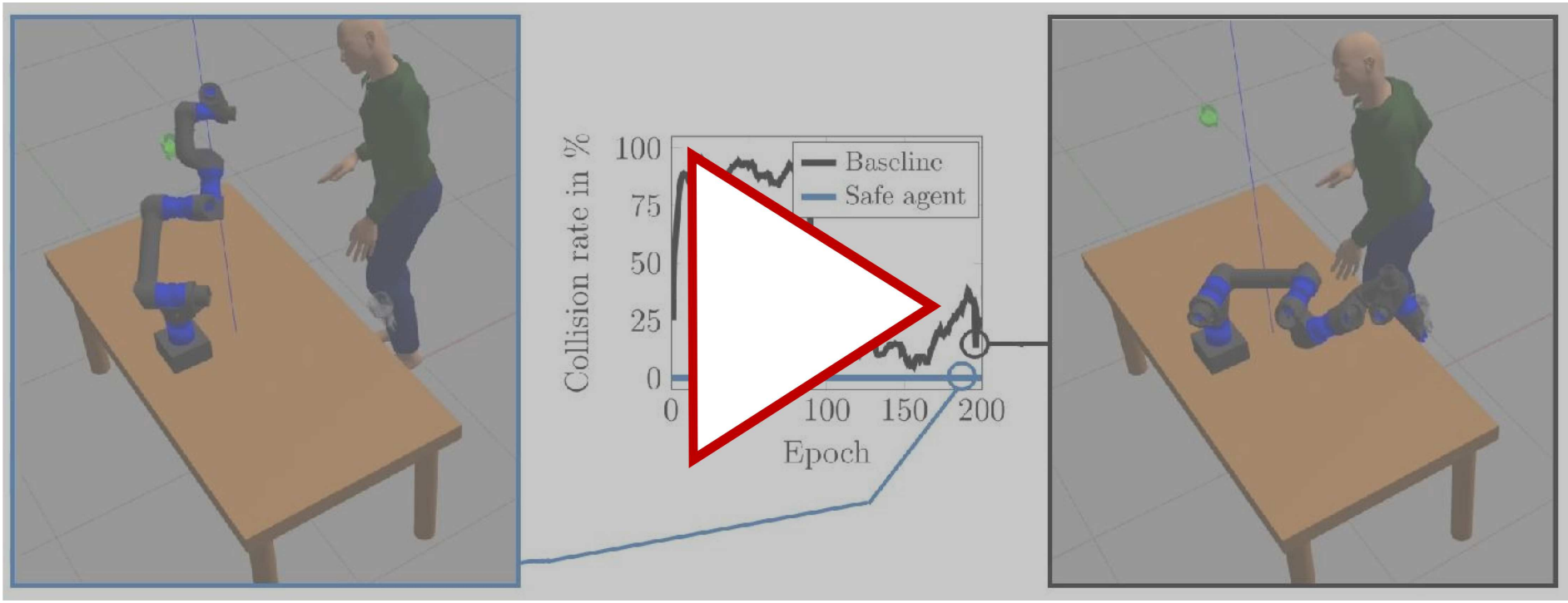
Unsafe mode



Safe mode

Results Video

Not relevant for the exam



Learning for Power Systems

Not relevant for the exam

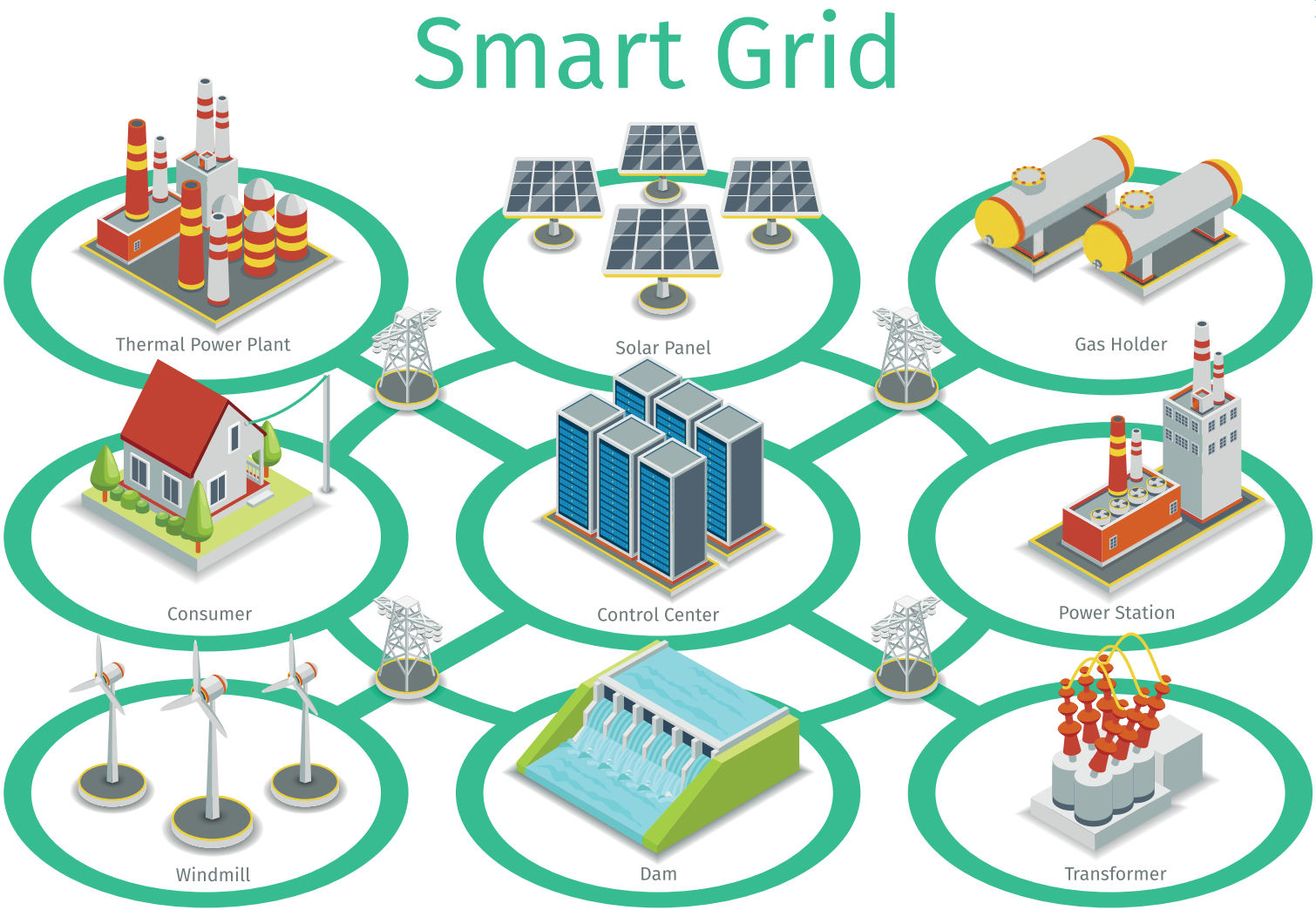


Image by macrovector on Freepik

Learning for Power Systems - Flexible Grid Management

Not relevant for the exam

Motivation

- Increasing volatility of power supply (wind, PV) requires increasing **flexibility on the demand side**.
- Residential PV and battery systems as well as heat pumps and electric vehicles (EV) allow customers to become **prosumers** who can provide some of the required flexibility.
- Individual prosumers **need to be coordinated** to avoid undesired behavior.

Our research

- Smart pricing for ancillary services of EVs.
- Multi-agent RL for coordinating demand response in communities.
- Safeguarding RL for power systems.

Learning for Power Systems - Forecasting

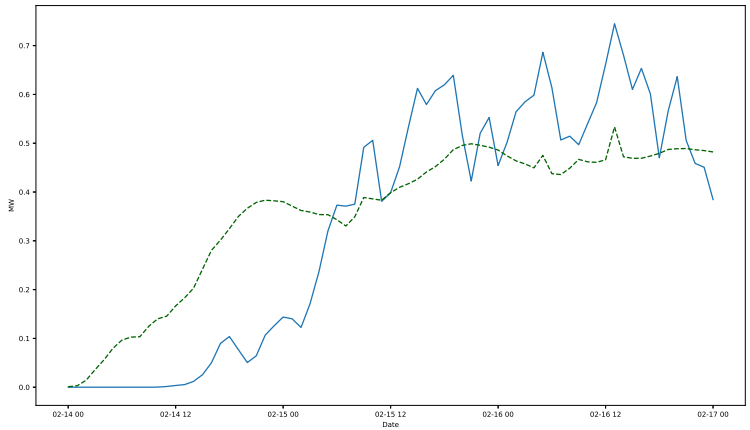
Not relevant for the exam

Motivation

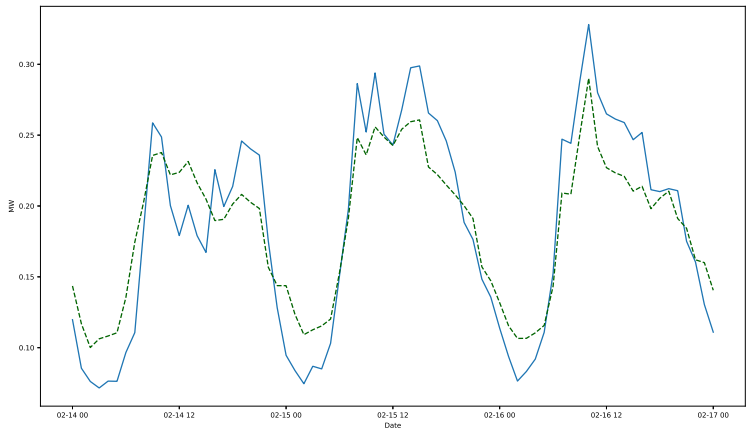
- Efficient power system operation requires accurate forecasts of demand, generation and prices.
- Non-learning-based methods are either computationally expensive or often inaccurate.

State of the art

Hybrid models combining a neural network (e.g. MLP, CNN, RNN, LSTM) with a numerical weather model.



Exemplary wind power generation forecast (dotted line)



Exemplary power demand forecast

Sounds Interesting? Work With Us!

Not relevant for the exam

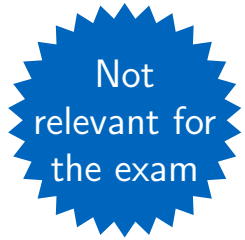
We offer

- Practical course "Machine Learning for Smart Grids"
- Practical course "Reinforcement Learning for Modular Robots"
- Practical course "Motion planning for the autonomous vehicle EDGAR"
- Topics in the seminar "Cyber-physical systems"
- Thesis topics (BA, MA)
- Guided research

Get in touch!

Hannah Markgraf	hannah.markgraf@tum.de	power systems
Michael Eichelbeck	michael.eichelbeck@tum.de	power systems
Jakob Thumm	jakob.thumm@tum.de	robotics
Jonathan Külz	jonathan.kuelz@tum.de	robotics
Xiao Wang	xiao.wang@tum.de	autonomous driving
Hanna Krasowski	hanna.krasowski@tum.de	autonomous driving
Eivind Meyer	eivind.meyer@tum.de	autonomous driving
Luis Gressenbuch	luis.gressenbuch@tum.de	autonomous driving

Beyond Decision Tree Learning and Reinforcement Learning



So far, we have only focused on decision trees and reinforcement learning. Machine Learning has much more to offer, e.g.,

- classification with linear models;
- artificial neural networks;
- clustering techniques;
- support vector machines;
- ensemble learning, etc.

These topics are covered in machine learning lectures at our department.

Summary

- Learning is needed for unknown environments and lazy designers.
- Types of learning: Unsupervised learning, reinforcement learning, supervised learning.
- In supervised learning, one tries to learn a function $y = h(x)$ from input-output pairs.
- It is crucial to find a hypothesis that agrees well with the examples. **Ockham's razor** maximizes a combination of consistency and simplicity.
- **Decision trees** can represent all Boolean functions. The information-gain heuristic efficiently finds simple decision trees.
- **Reinforcement learning** makes it possible that an agent improves its performance without data.