RL LE1 VT2026

Introduction and Tabular

Fredrik Heintz

Dept. of Computer Science
Linköping University

fredrik.heintz@liu.se
@FredrikHeintz

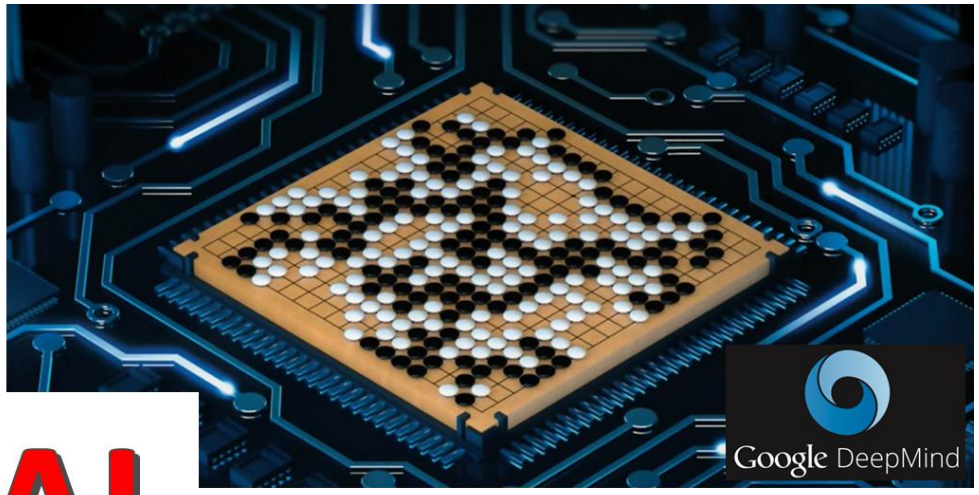
LINKÖPING UNIVERSITY

Outline:

- **Positioning Reinforcement Learning in AI**
- **Introduction to Reinforcement learning**
- **Course Overview**
- **Value Iteration and Policy Iteration**
- **Tabular Methods (Q-Learning, SARSA)**
- **Exploration and Exploitation**

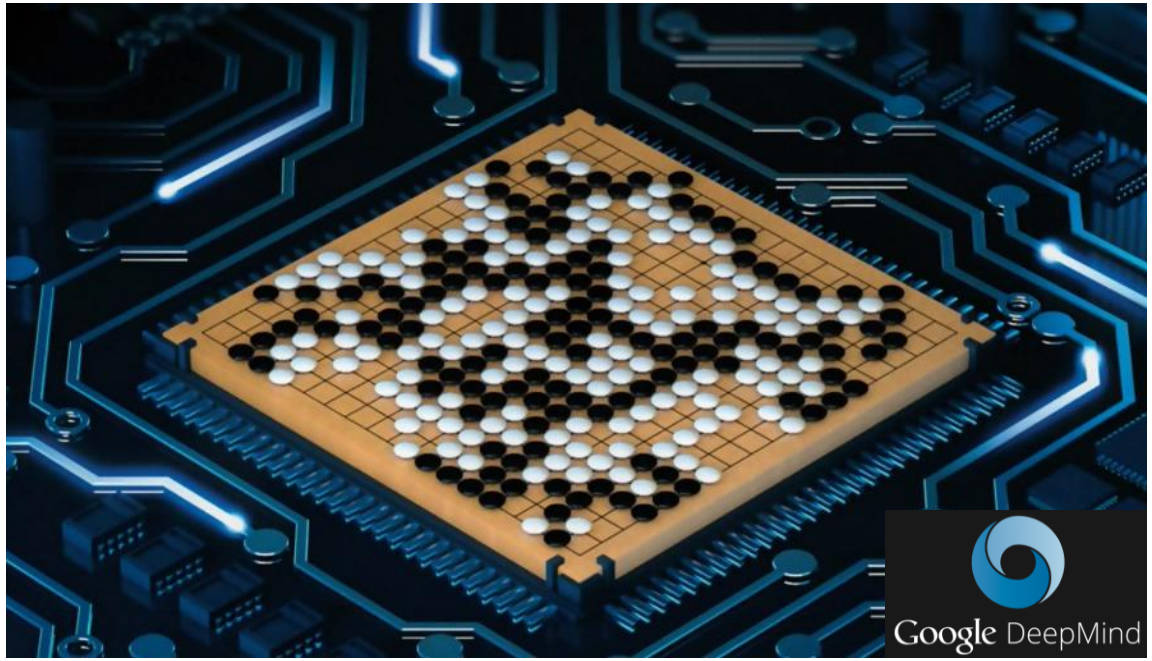
AI

- Classification
- Prediction
- Generation
- Acting



Acting

- Playing games – AlphaGo, StarCraft
- Manipulation – Rubrik's Cube
- Flying – Stanford helicopter project
- Moving – Learning to walk



Sequential Decision Making (under Uncertainty)

- Optimal Control / Motion Planning – Given a model of the system and the environment compute the optimal behaviour
- Automatic (Task) Planning – Given a model of the world and a goal find the optimal plan
- Planning under Uncertainty – Given a probabilistic model of the world find the optimal plan given the uncertainty modelled
- Reinforcement Learning – Given an objective learn to maximise it through interaction

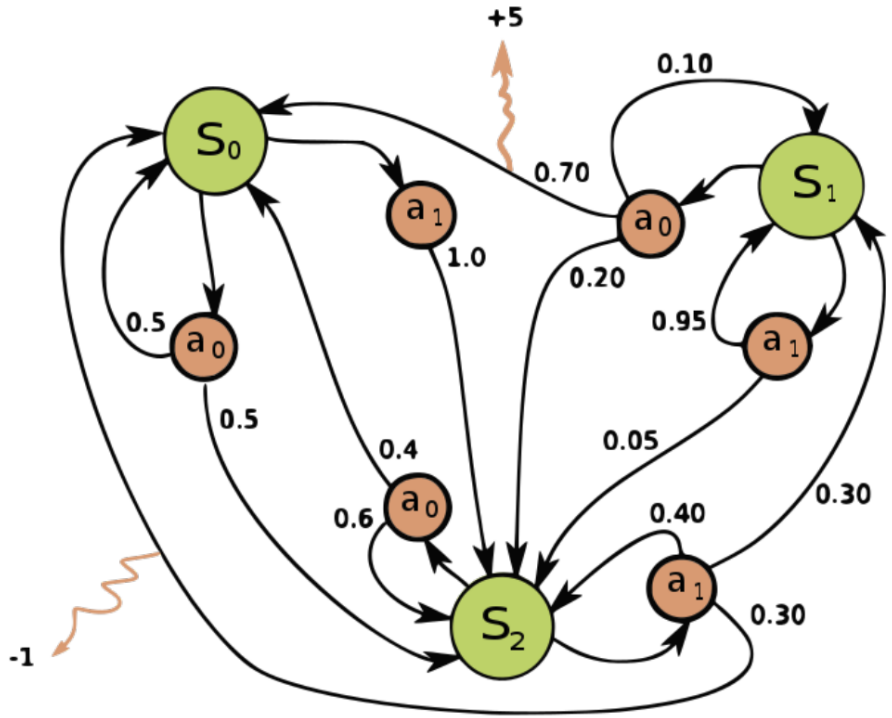
Markov Decision Processes (MDP)

Assume:

- finite set of states S, finite set of actions A
- at each discrete time agent observes state $s_t \in S$ and chooses action $a_t \in A$
- then receives immediate reward r_t
- and state changes to s_{t+1}
- Markov assumption: $s_{t+1} = \delta(s_t, a_t)$ and $r_t = r(s_t, a_t)$
  - i.e. r_t and s_{t+1} depend only on current state and action
  - functions δ and r may be non-deterministic
  - functions δ and r not necessarily known to the agent

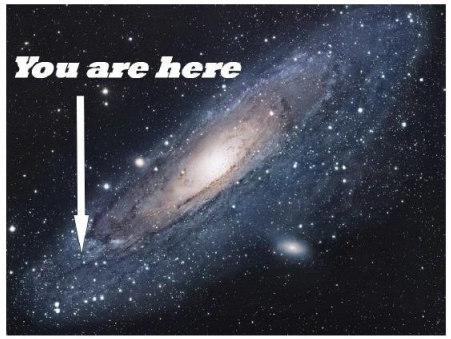
MDP Example

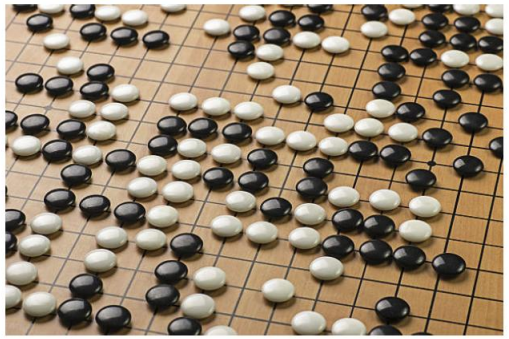
- S - State
- A - Action
- T - probability of Transitioning
- R- Reward (can be positive and negative)
- γ - Discount factor

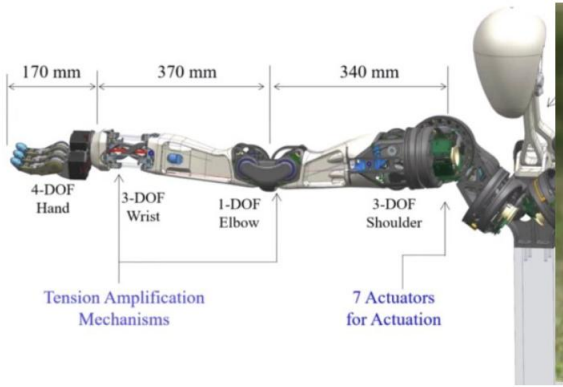


State

- Uniquely represent the state of the environment at time *t*
  - location on a map
  - pieces on a board
  - angles of joints
  - pixel values in a grid







Action

- state $s \rightarrow$ action $a \rightarrow$ state s'
- Discrete action:
  - a small integer number
  - move pawn e2 to e4
- Continuous action:
  - bet $1234,56
  - move joint A to 56,7 degrees
- Discrete policy $\pi(s) \rightarrow a$
- Stochastic policy $\pi(a|s) \rightarrow$ probability distribution over actions

Transition

- state $s \rightarrow$ action $a \rightarrow$ state s'
- $T(s, a, s')$ is the probability that action a in state s will transition to state s'
- Transitions can be deterministic or non-deterministic
- In $s \rightarrow a \rightarrow s'$
  - the $s \rightarrow a$ part is chosen by the agent (policy)
  - the $a \rightarrow s'$ part is chosen by the environment
- T is known by the environment, not by the agent

Transition Model

- Model-free methods:
  - T is known by Environment only
  - For example: Q-learning
- Model-based methods:
  - Agent has local (approximation of) T
  - For example: Dyna

Trajectory

- Episodic problems have an end
- Continuous problems continue for ever
- Trajectory/Trace/Episode is the sequence of state/action/reward from start to finish

$$\tau_t^n = \{s_t, a_t, r_t, s_{t+1}, .., a_{t+n}, r_{t+n}, s_{t+n+1}\}$$

Reward

- $R(s, a, s')$ is the Reward received after action a transitions from state s to state s'
- $R(\tau)$ is the Return: the cumulative reward of a trace
- $V^{\pi}(s)$ is the state Value: the expected cumulative reward of a state s for following the policy π from s
- $Q^{\pi}(s, a)$ is the state-action Value: the expected cumulative reward of a state s for following action a from state s and then the policy π from s'

γ (Gamma)

- γ (gamma) is the discount factor, discounting the importance of future rewards
- Especially important in continuous problems
- Sometimes ignored (γ=1) in episodic problems

Functions

- Value $V(s)$
- Action Value $Q(s, a)$
- Policy $\pi(s)$
- It may help to think of these functions as arrays that can be updated

The Value Function

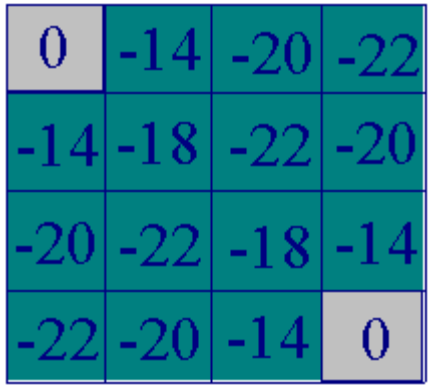
- The Bellman equation recursively defines the value of a state (assuming transition function P and policy π are given).
- Discounted future reward:

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma \sum_{s'} P(s'|s, \pi(s)) V^{\pi}(s')$$

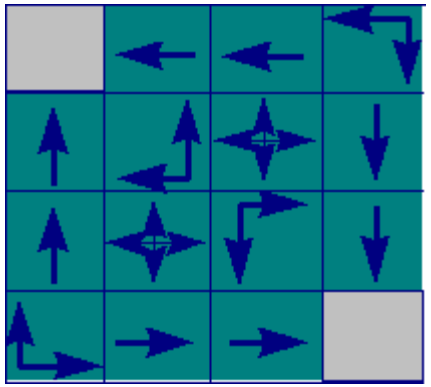
- Needs Reward R, Policy π, and Transition P.
- It is nice to have this recursive equation, but unfortunately, we typically do not have the transition function.

Value Function - Example

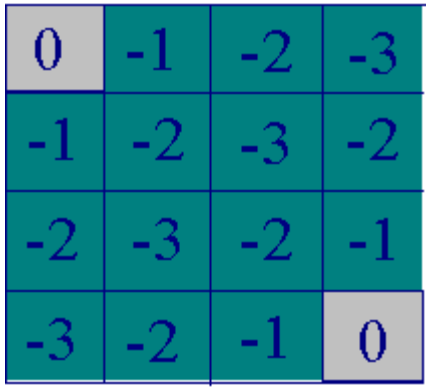
A minimum time to goal world



Value function for random movement



Optimal policy

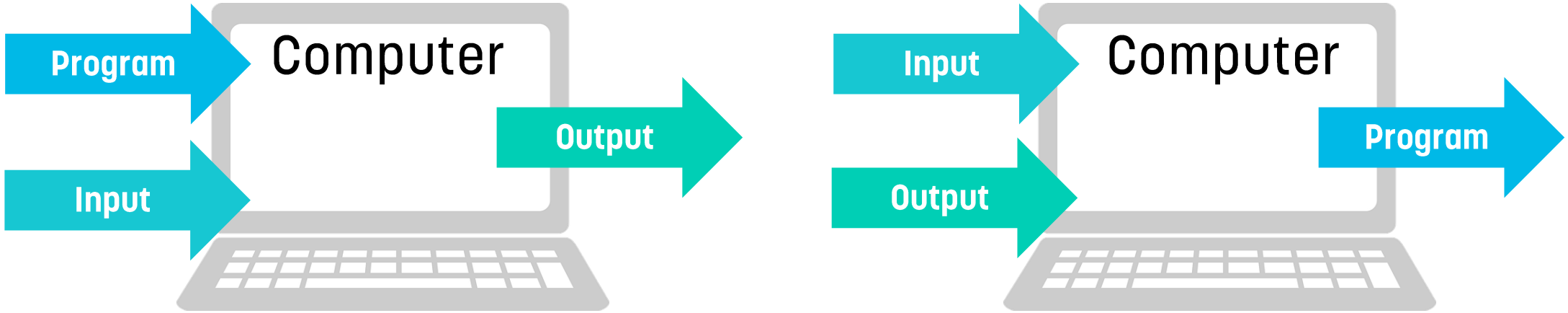


Optimal value function

Markov Models

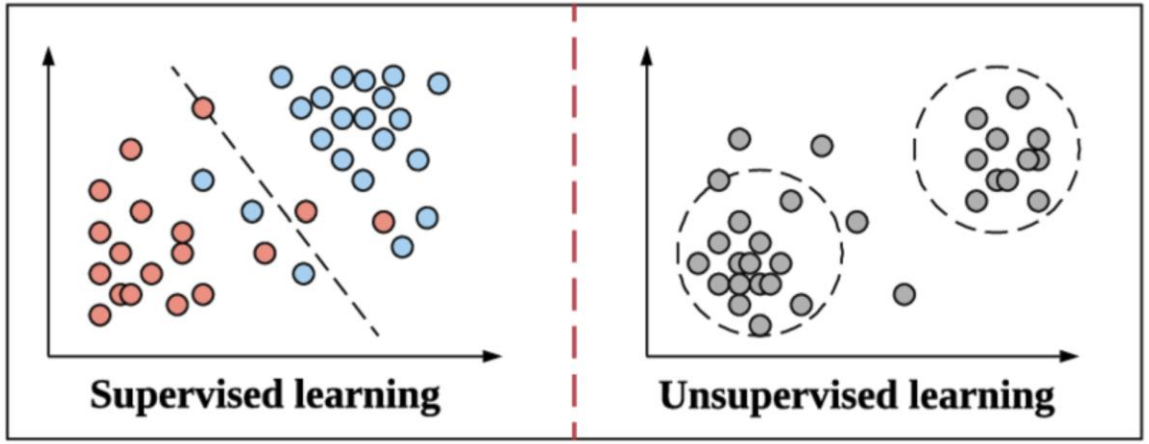
	No Agents	Single Agent	Multiple Agents
State Known	Markov Chain	Markov Decision Process (MDP)	Markov Game (a.k.a. Stochastic Game)
State Observed Indirectly	Hidden Markov Model (HMM)	Partially-Observable Markov Decision Process (POMDP)	Partially-Observable Stochastic Game (POSG)

Machine Learning

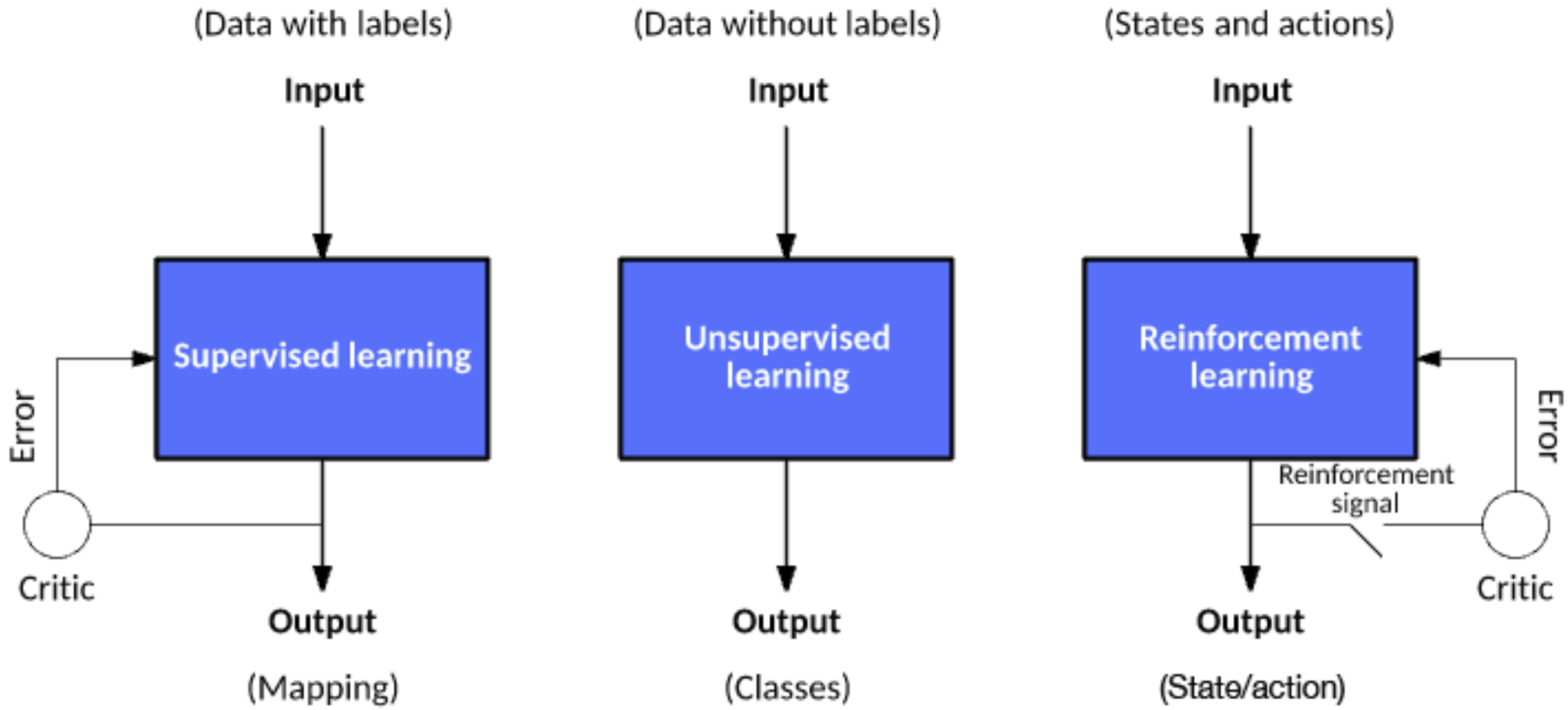


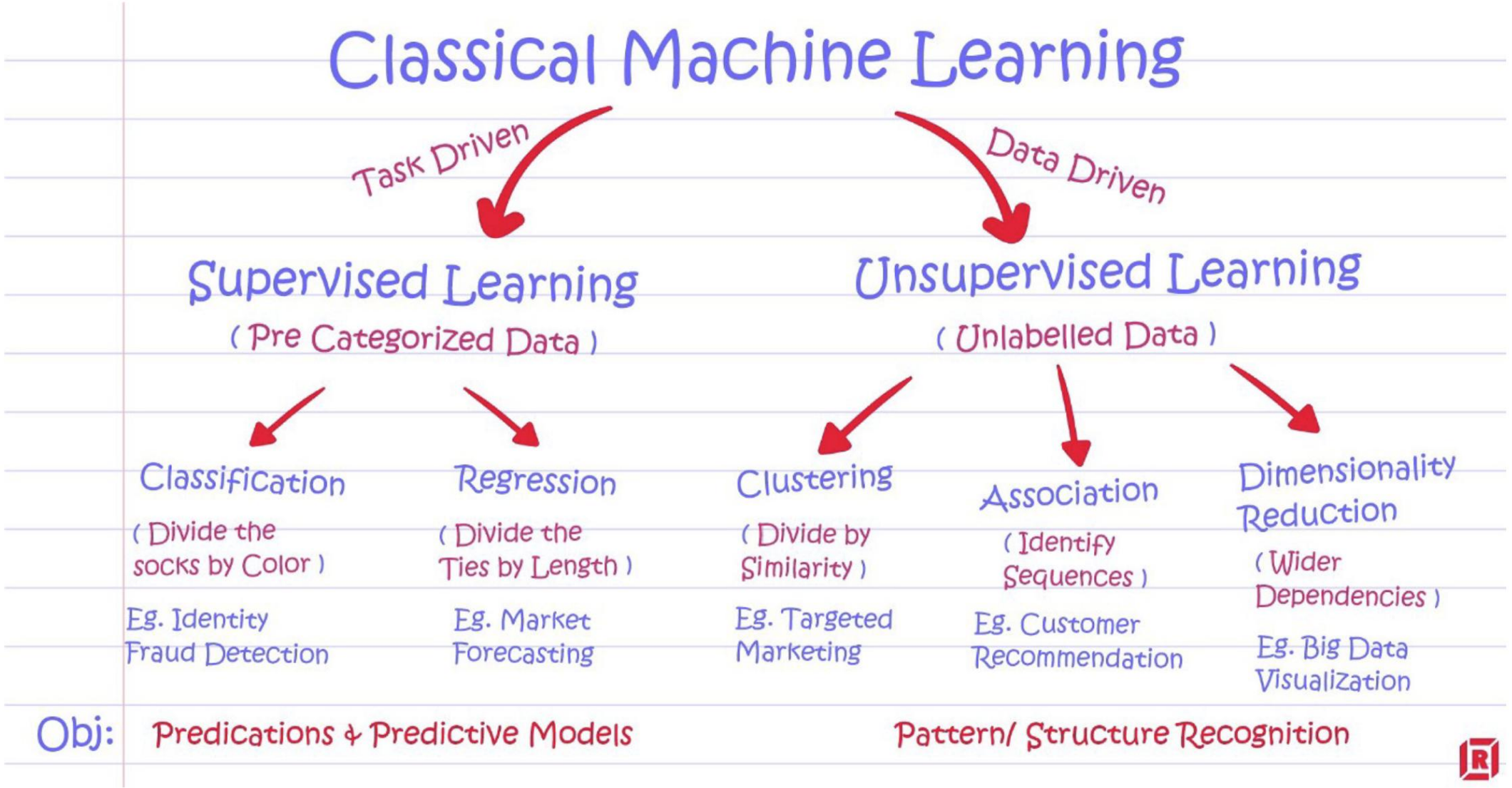
Machine Learning Approaches

- Supervised Learning – Learning from (input, output) examples
- Self-Supervised Learning – Learning from examples where labels deduced
- Semi-Supervised Learning – Learning from partially labelled examples
- Unsupervised Learning – Learning the structure of unlabelled examples
- Reinforcement Learning – Learning from interaction

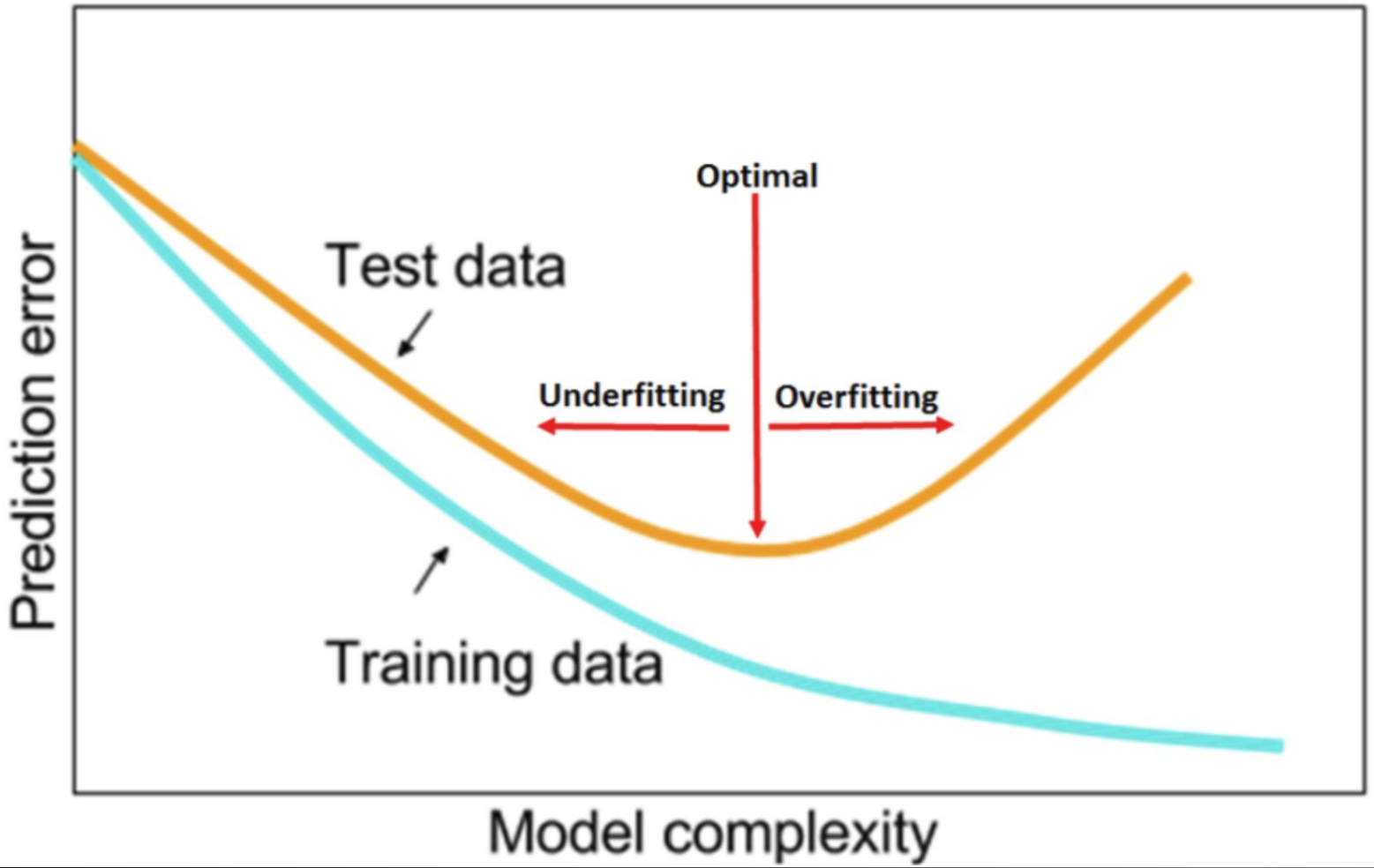


Machine Learning

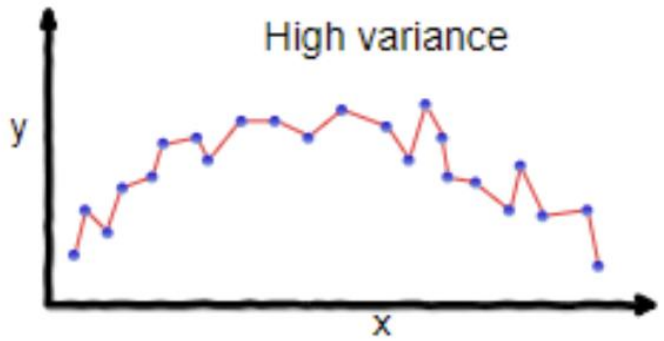



Classical Machine Learning
Task Driven
Data Driven
Supervised Learning
(Pre Categorized Data)
Unsupervised Learning
(Unlabelled Data)
Classification
(Divide the socks by Color)
Eg. Identity Fraud Detection
Regression
(Divide the Ties by Length)
Eg. Market Forecasting
Clustering
(Divide by Similarity)
Eg. Targeted Marketing
Association
(Identify Sequences)
Eg. Customer Recommendation
Dimensionality Reduction
(Wider Dependencies)
Eg. Big Data Visualization
Obj:
Predications & Predictive Models
Pattern/ Structure Recognition

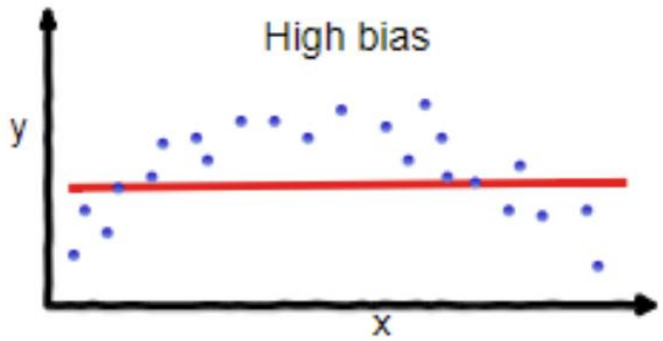
Generalisation



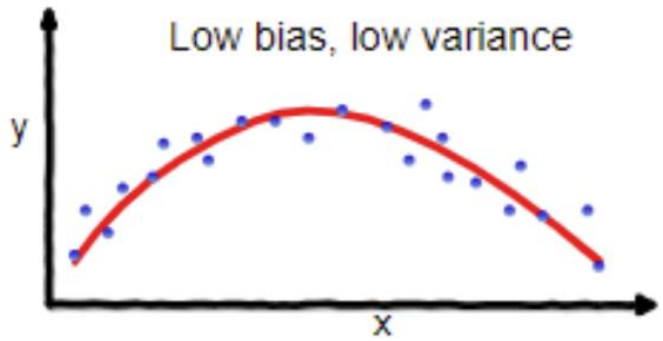
Bias-Variance



overfitting

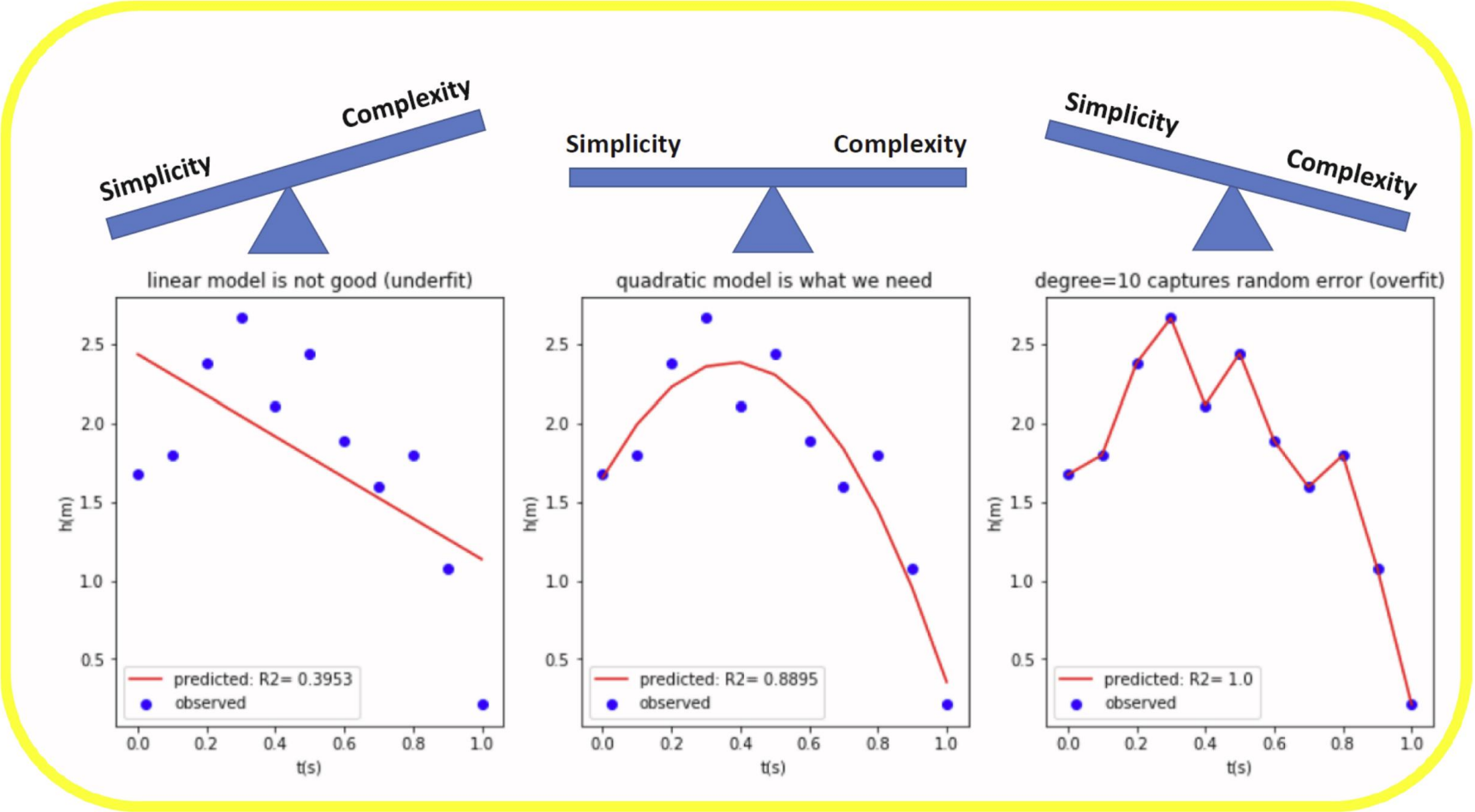


underfitting



Good balance

Balancing Simplicity and Complexity



Learning and Sequential Decision Making

- Representation Learning – Learning a world model from observations
- Learning Causal Relations – Learning how the world works
- Imitation Learning – Learning how to act from observing actions
- Reinforcement Learning – Learning how to act through interactions
- Inverse Reinforcement Learning – Learning the reward function from observations

Introduction to Reinforcement Learning

Reinforcement Learning: Key Concepts

Agent: takes actions.

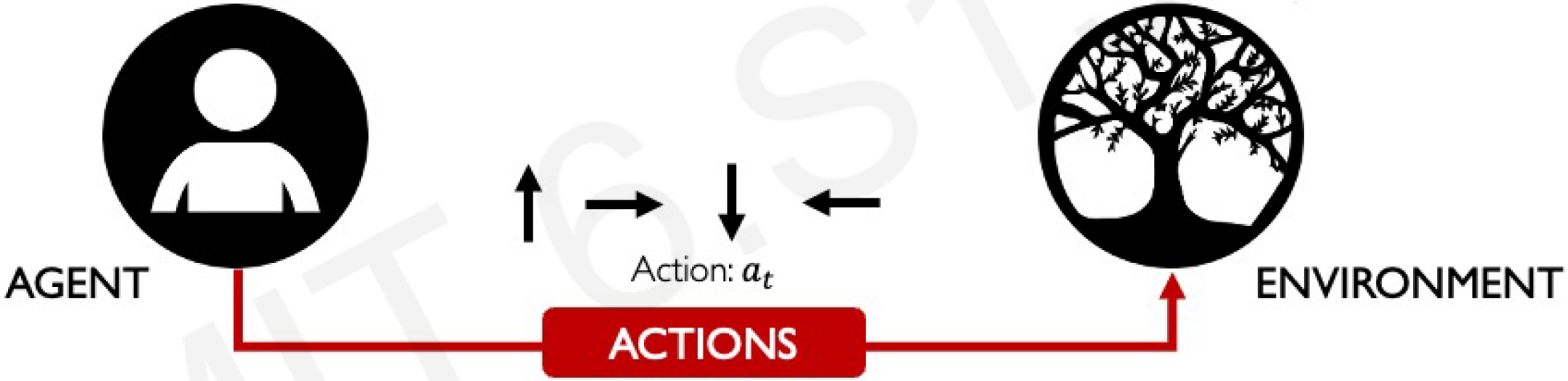
Reinforcement Learning: Key Concepts

AGENT

ENVIRONMENT

Environment: the world in which the agent exists and operates.

Reinforcement Learning: Key Concepts



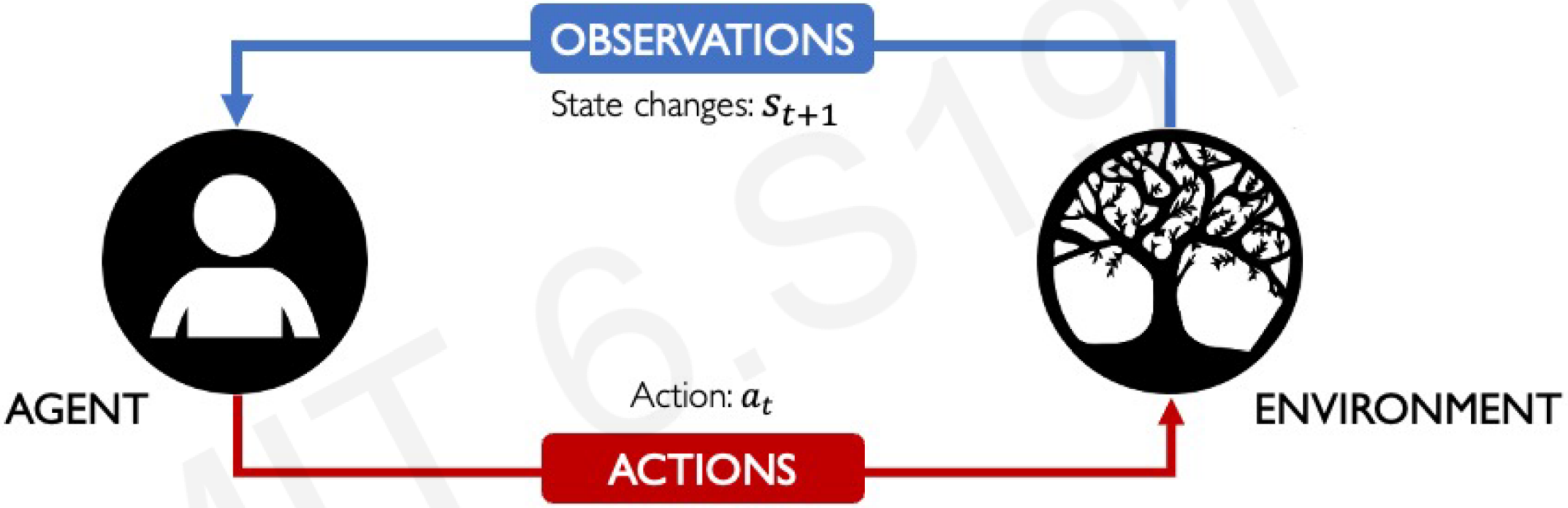
Action: a move the agent can make in the environment.

Action space A: the set of possible actions an agent can make in the environment

Reinforcement Learning: Key Concepts

Observations: of the environment after taking actions.

Reinforcement Learning: Key Concepts

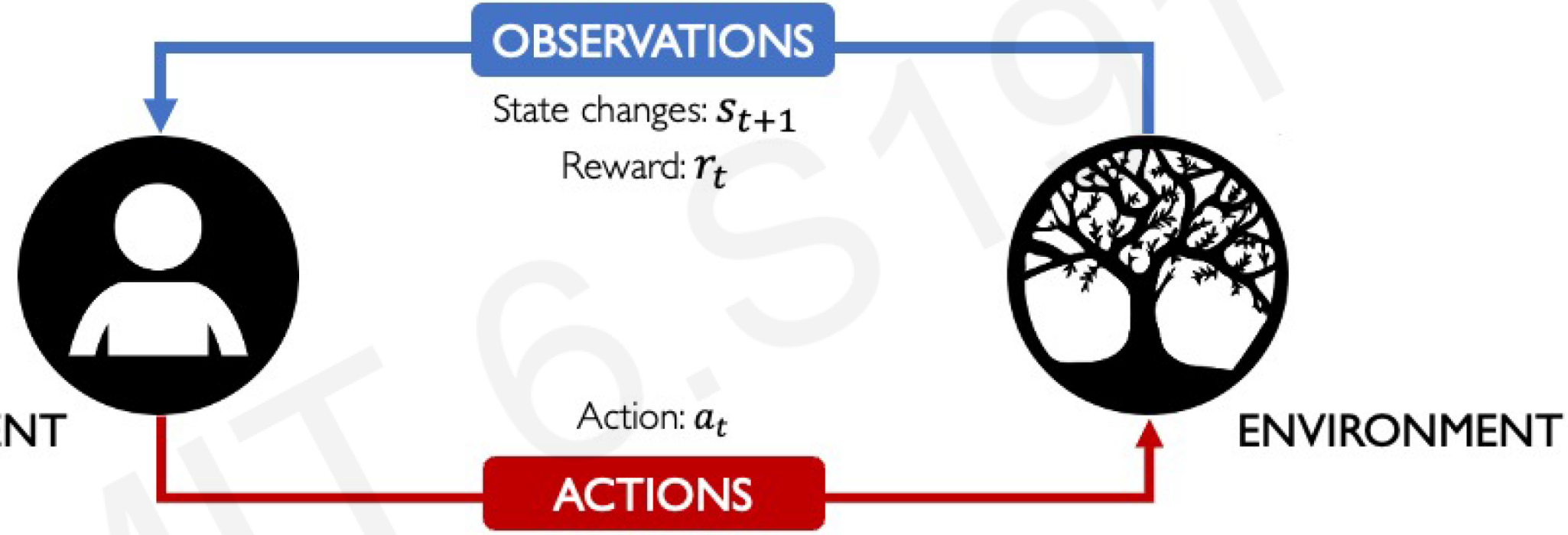


State: a situation which the agent perceives.

Reinforcement Learning: Key Concepts

Reward: feedback that measures the success or failure of the agent's action.

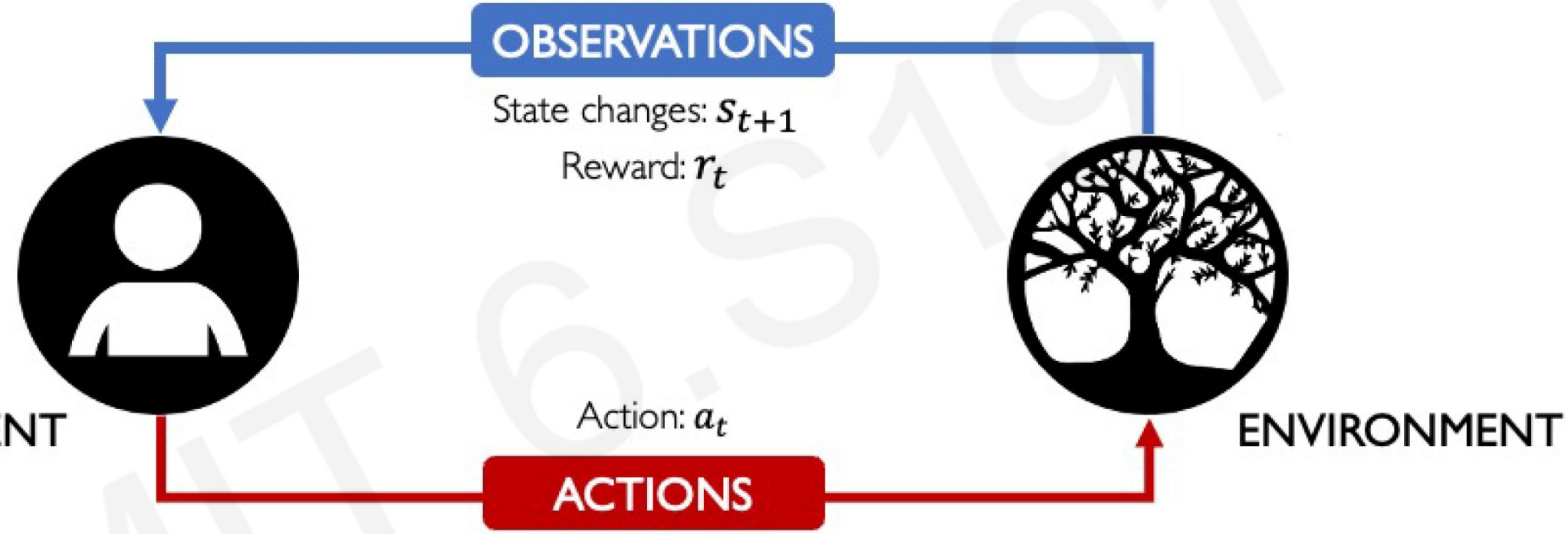
Reinforcement Learning: Key Concepts



Total Reward (Return)

$$R_t = \sum_{i=t}^{\infty} r_i$$

Reinforcement Learning: Key Concepts



Total Reward (Return)

$$R_t = \sum_{i=t}^{\infty} r_i = r_t + r_{t+1} \ldots + r_{t+n} + \cdots$$

Reinforcement Learning: Key Concepts

Discounted Total Reward (Return)

$$R_t = \sum_{i=t}^{\infty} \gamma^i r_i$$

Reinforcement Learning: Key Concepts

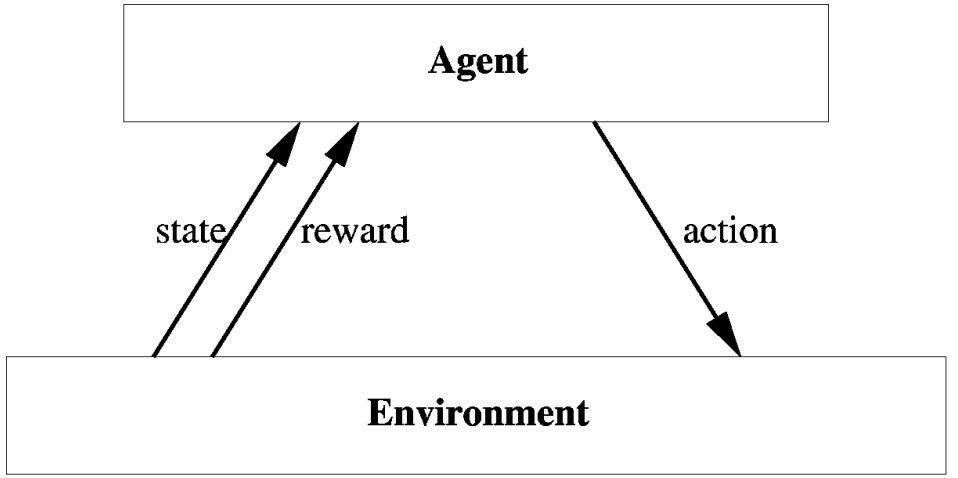
Discounted Total Reward (Return)

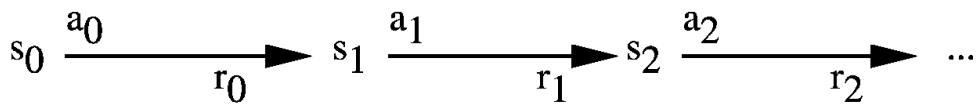
$$R_t = \sum_{i=t}^{\infty} \gamma^i r_i = \gamma^t r_t + \gamma^{t+1} r_{t+1} \ldots + \gamma^{t+n} r_{t+n} + \cdots$$

γ: discount factor; $0 < \gamma < 1$

A Reinforcement Learning Problem

- The environment
- The reward function *r(s,a)*
  - Pure delay reward and avoidance problems
  - Minimum time to goal
  - Games
- The value function *V(s)*
  - Policy π: $S \rightarrow A$
  - Value $V^{\pi}(s) := \Sigma_i \gamma^i r_{t+i}$
- Find the optimal policy π^* that maximizes $V^{\pi^*}(s)$ for all states *s*.



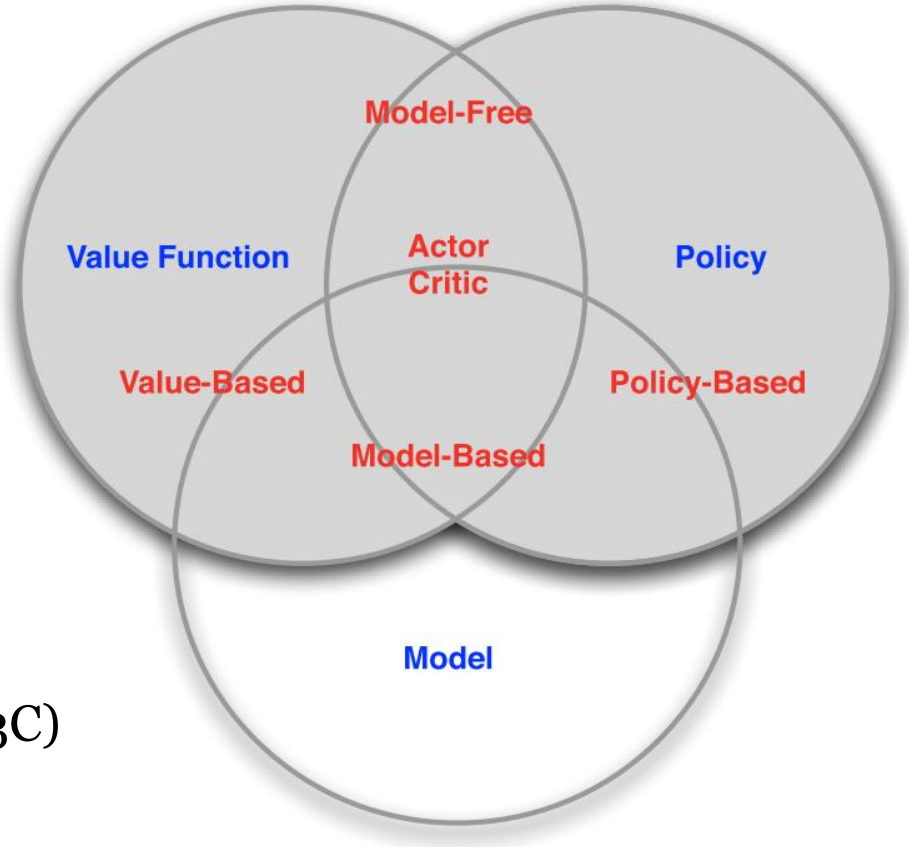


Goal: Learn to choose actions that maximize

$r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots$, where $0<\gamma<1$

Reinforcement Learning Approaches

- Value-Based:
  - Learn value function
  - Implicit policy (e.g. greedy selection)
  - Example: Deep Q Networks (DQN)
- Policy-Based:
  - No value function
  - Learn explicit (stochastic) policy
  - Example: Stochastic Policy Gradients
- Model-Based:
  - Learn transition model
  - Implicit policy
  - Example: Dreamer
- Actor-Critic:
  - Learn value function
  - Learn policy using value function
  - Example: Asynchronous Advantage Actor Critic (A3C)



Reinforcement Learning Overview

- Single|multiple agent
- single|multi task
- single|multi objective
- stationary|non-stationary
- partially|fully observable
- (constrained)
- discrete|continuous state
- discrete|continuous action
- solved exactly|approximately

Course Overview

Course Goal

- Understand the basic principles of reinforcement learning and deep reinforcement learning and how it differs from other AI/ML methods.
- Understand the theory and the algorithms behind common Deep RL algorithms such as DQN, TRPO, PPO, SAC, TD3.
- Implement and evaluate common Deep RL algorithms.
- Describe and apply metrics for analyzing and evaluating RL algorithms: e.g. regret, sample complexity, computational complexity, empirical performance, convergence, etc.
- Describe the exploration vs exploitation challenge and compare and contrast at least two approaches for addressing this challenge (in terms of performance, scalability, complexity of implementation, and theoretical guarantees).

Learning Outcomes

- The course gives an introduction to the field reinforcement learning. The aim is that students should be acquainted with different methods that are used for learning based on feedback.
- On completion of the course, you should be able to:
  - identify basic concepts, terminology, theories, models and methods in reinforcement learning
  - develop and systematically test basic methods in reinforcement learning
  - evaluate different reinforcement learning algorithms experimentally as well as interpret and document results of experimental studies
  - have a broad knowledge to be able to read and profit by literature in the area

Course Lectures

- 1/4 LE1 **Introduction to reinforcement learning** [Fredrik]
- 8/4 LE2 **Value-based and Policy-based Deep RL** [Amath]
- 15/4 LE3 **Model-based and Actor-Critic RL** [Fredrik]
- 22/4 LE4 **RL for LLMs** [Amath]
- 29/4 LE5 **Multi-Objective RL** [Fredrik]
- 6/5 LE6 **Multi-agent RL** [Amath]
- 13/5 LE7 **Advanced Topics** [Fredrik]
- 20/5 SE [Amath]

Examination

The examination consists of two components:

Module	ECTS	Description
LAB1	4 hp	5 lab assignments – group code submission
LAB2	2 hp	Individual written report

Both components are graded on a scale of **U / 3 / 4 / 5**. The final course grade is computed in two steps:

1. **LAB1 grade** = mean of the 5 individual lab grades (rounded to nearest integer)
2. **Final grade** = mean of LAB1 grade and LAB2 report grade (rounded to nearest integer)

Examination - LAB1

- Labs may be completed **individually or in groups of 2-3**. Please sign up in **WebReg** before April 14.
- All group members must be able to explain and defend all submitted code and results.
- Each lab consists of two parts:
  - **Part A - Implementation:** Build the algorithm(s) from scratch in PyTorch following the TODO markers in the starter notebook.
  - **Part B - Experiments:** Run systematic ablation studies and comparisons, produce plots, and write a short analysis.

Lab	Topic	Core Methods	Environment(s)	Submission Deadline
Lab 1	Value-Based Deep RL	DQN, Double DQN, Dueling DQN	CartPole-v1, LunarLander-v3	Fri 17 Apr, 23:59
Lab 2	Policy Gradient	REINFORCE (with/without baseline), PPO	CartPole-v1	Fri 1 May, 23:59
Lab 3	Actor-Critic	A2C (with GAE), SAC	LunarLanderContinuous-v3	Fri 8 May, 23:59
Lab 4	Model-Based Deep RL	Dyna-Q (neural WorldModel), MCTS	CliffWalking-v1	Fri 15 May, 23:59
Lab 5	Multi-Agent Deep RL	MAPPO, MADDPG	simple_spread_v3 (PettingZoo)	Fri 29 May, 23:59

Examination – LAB2

LAB2 - Individual Written Report

The report is **individual** - every student submits their own report regardless of lab group.

Each student chooses **one** of the following two options:

Option A - Lab Summary Report

Write a report summarising your work and findings across **all 5 labs**.

The report must be **at most 6 pages** (excluding figures and references) and cover:

1. **Overview** - briefly describe each algorithm implemented and the environment it was tested on
2. **Key results** - one or two main findings per lab (learning curves, ablation conclusions)
3. **Comparative analysis** - compare methods across labs (e.g., sample efficiency, stability, scalability)
4. **Reflection** - what worked well, what was challenging, and what you learned

Option B - Paper Presentation: RL for LLMs

Select a published research paper on the application of **reinforcement learning to large language models** (e.g., RLHF, PPO for alignment, GRPO, reward modelling).

The report must be **at most 6 pages** (excluding figures and references) and include:

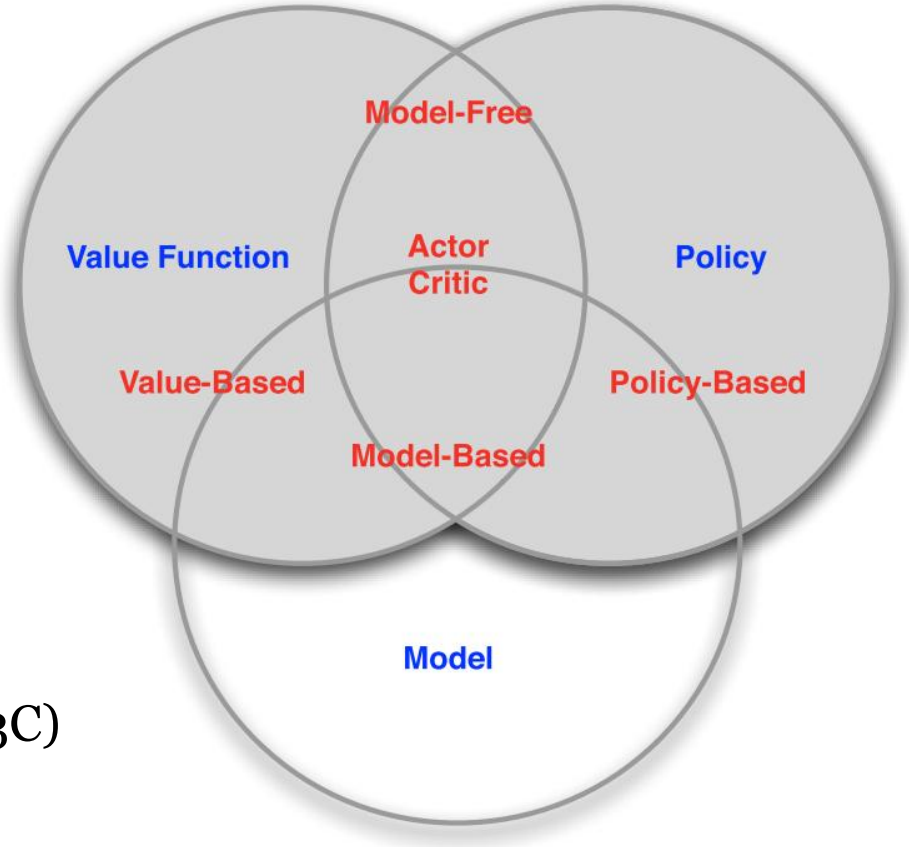
1. **Paper summary** - problem setting, proposed method, key contributions
2. **Connection to course content** - which algorithms from the labs are used or related
3. **Critical analysis** - strengths, limitations, and open questions
4. **References** - at least 3 published papers from established venues

The paper must be approved by the course responsible **before** you start writing. Send a short paragraph proposal with the paper title to **amath.sow@liu.se**.

Value Iteration and Policy Iteration

Reinforcement Learning Approaches

- Value-Based:
  - Learn value function
  - Implicit policy (e.g. greedy selection)
  - Example: Deep Q Networks (DQN)
- Policy-Based:
  - No value function
  - Learn explicit (stochastic) policy
  - Example: Stochastic Policy Gradients
- Model-Based:
  - Learn transition model
  - Implicit policy
  - Example: Dreamer
- Actor-Critic:
  - Learn value function
  - Learn policy using value function
  - Example: Asynchronous Advantage Actor Critic (A3C)



Value Iteration

- *Value Iteration* finds the optimal value function V* by solving the Bellman equations iteratively.
- Uses dynamic programming to maintain a value function V that approximates the optimal value function V*, iteratively improving V until it converges to V*.

Algorithm – Value Iteration

Input: MDP $M = \langle S, s_0, A, P_a(s' \mid s), r(s,a,s') \rangle$

Output: Value function V

Set V to arbitrary value function; e.g., $V(s) = 0$ for all s

Repeat

$\Delta \leftarrow 0$

For each $s \in S$

$$\underbrace{V'(s) \leftarrow \max_{a \in A(s)} \sum_{s' \in S} P_a(s' \mid s) \left[r(s,a,s') + \gamma V(s') \right]}_{\text{Bellman equation}}$$

$\Delta \leftarrow \max(\Delta, |V'(s) - V(s)|)$

$V \leftarrow V'$

Until $\Delta \leq \theta$

https://gibberblot.github.io/rl-notes/single-agent/value-iteration.html

Temporal Difference Learning [Sutton]

- Solution method that samples from environment, estimating the policy, when no transition probabilities are given

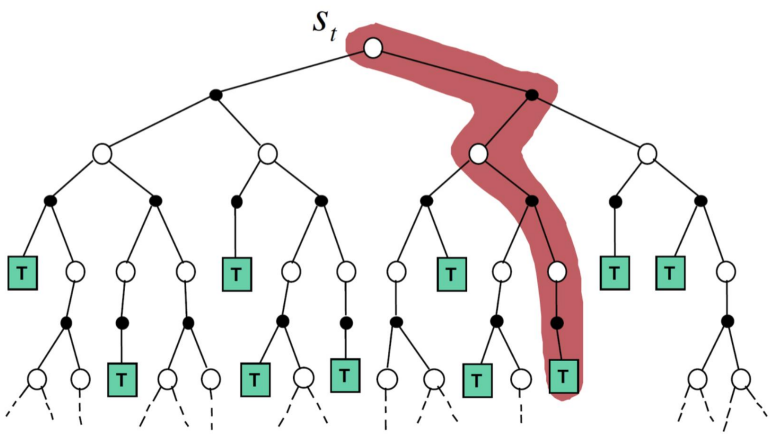
$$V(s) \leftarrow V(s) + \alpha[R' + \gamma V(s') - V(s)]$$

- Gamma is discount rate, Alpha is learning rate
- TD learns directly from episodes of experience
- TD is model-free, no knowledge of transitions or rewards
- TD learns from incomplete episodes by *bootstrapping*
- TD improves its guess for each iteration

Computing the Value Function

Monte Carlo

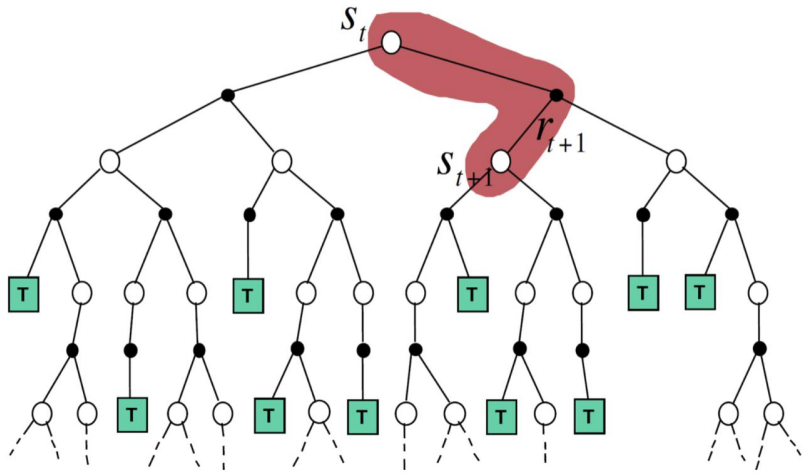
$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t - V(S_t)\right)$$



In Monte-Carlo we are basically traversing one random path of states which eventually leads to a terminating state. Hence, it will traverse through the **depth** and end with a terminating state.

Temporal Difference

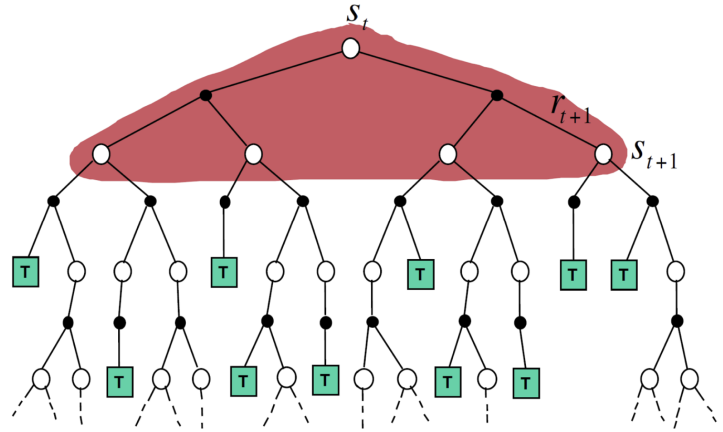
$$V(S_t) \leftarrow V(S_t) + \alpha\left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right)$$



In TD, we only look one step ahead and then estimate the rest. That is R_{t+1} + gamma*($V(S_{t+1})$.

Dynamic Programming

$$V(S_t) \leftarrow \mathbb{E}_\pi\left[R_{t+1} + \gamma V(S_{t+1})\right]$$

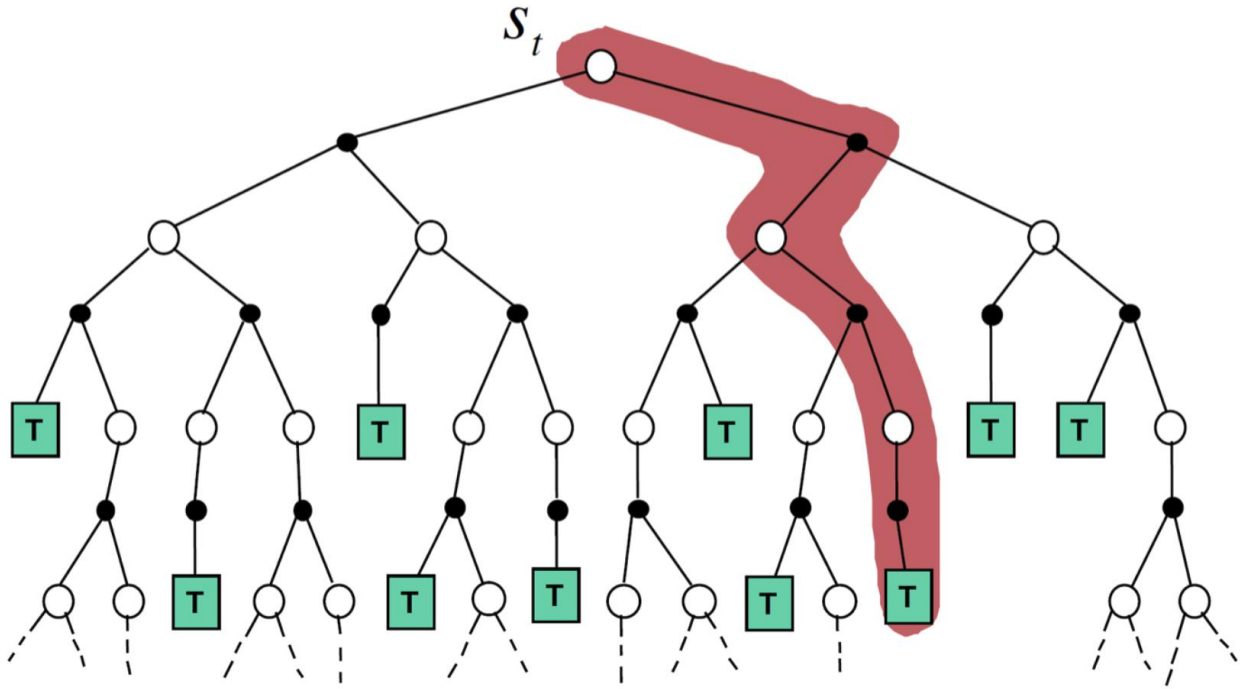


In DP, we used to consider **all possible states** one level ahead, i.e the entire breadth of level+1.

As opposed to this, in MC and TD we are only considering a limited space.

Monte Carlo Backup

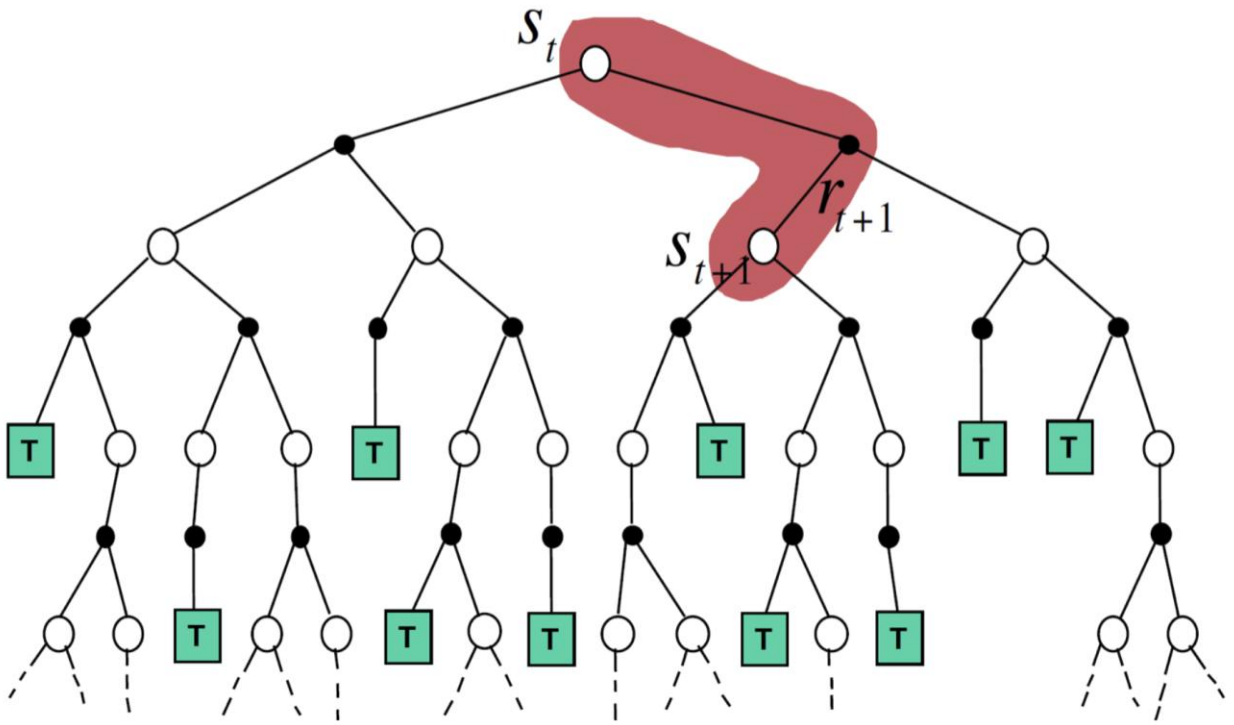
$$V(S_t) \leftarrow V(S_t) + \alpha\,(G_t - V(S_t))$$



In Monte-Carlo we are basically traversing one random path of states which eventually leads to a terminating state. Hence, it will traverse through the **depth** and end with a terminating state.

Temporal Difference

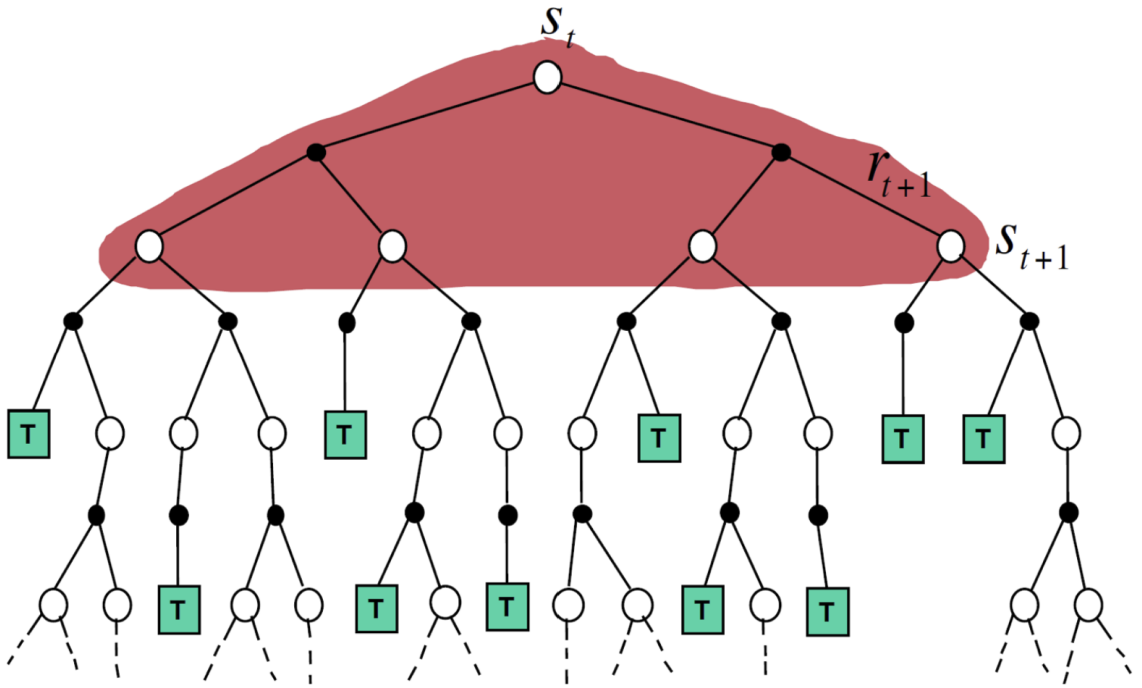
$$V(S_t) \leftarrow V(S_t) + \alpha\left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right)$$



In TD, we only look one step ahead and then estimate the rest. That is R_{t+1} + gamma*($V(S_{t+1})$.

Dynamic Programming

$$V(S_t) \leftarrow \mathbb{E}_\pi \left[R_{t+1} + \gamma V(S_{t+1}) \right]$$



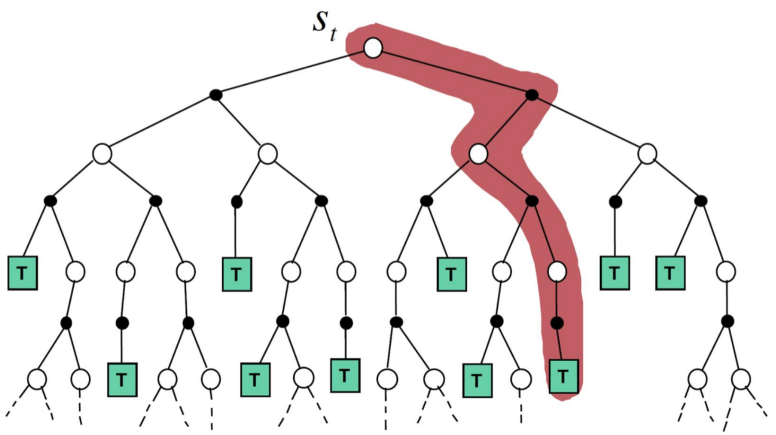
In DP, we used to consider **all possible states** one level ahead, i.e the entire breadth of level+1.

As opposed to this, in MC and TD we are only considering a limited space.

Computing the Value Function

Monte Carlo

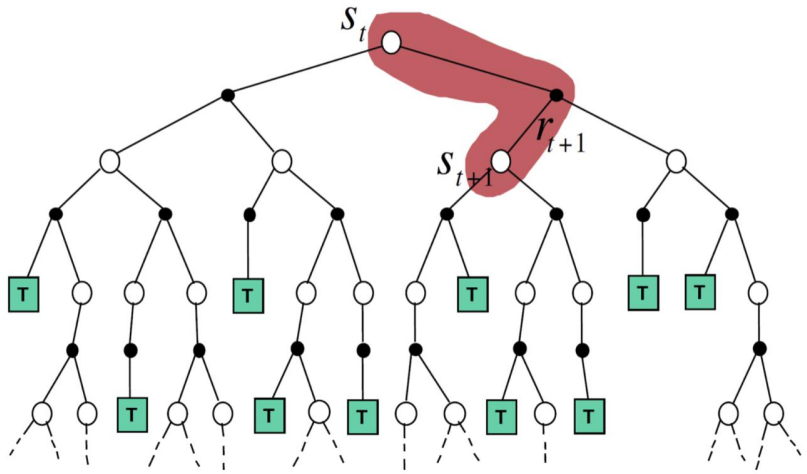
$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t - V(S_t)\right)$$



In Monte-Carlo we are basically traversing one random path of states which eventually leads to a terminating state. Hence, it will traverse through the **depth** and end with a terminating state.

Temporal Difference

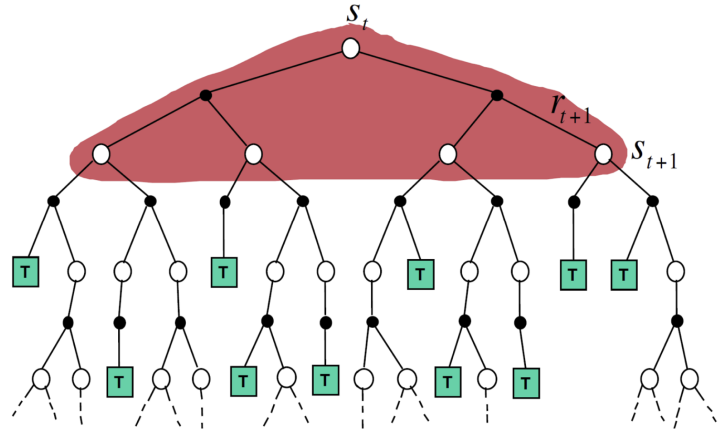
$$V(S_t) \leftarrow V(S_t) + \alpha\left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right)$$



In TD, we only look one step ahead and then estimate the rest. That is R_{t+1} + gamma*($V(S_{t+1})$.

Dynamic Programming

$$V(S_t) \leftarrow \mathbb{E}_\pi\left[R_{t+1} + \gamma V(S_{t+1})\right]$$

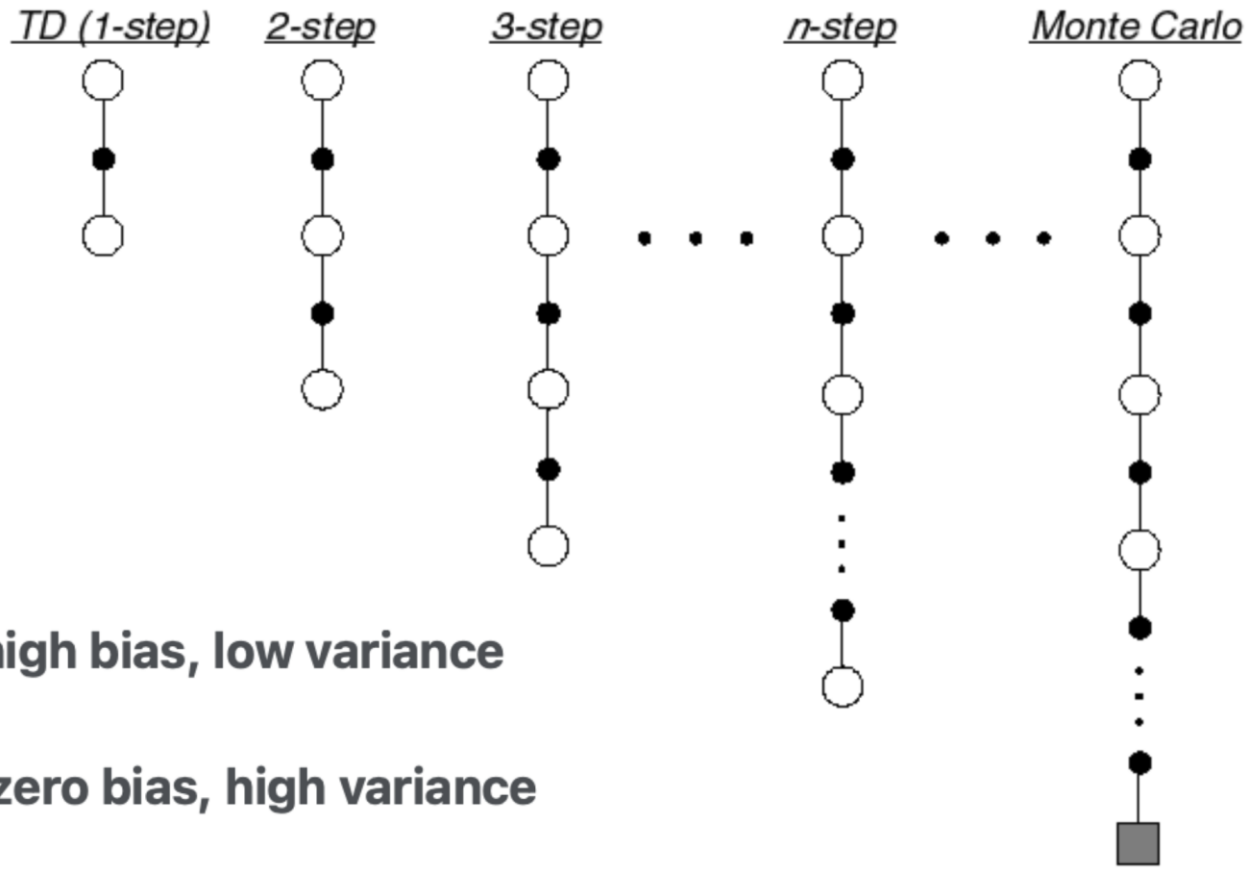


In DP, we used to consider **all possible states** one level ahead, i.e the entire breadth of level+1.

As opposed to this, in MC and TD we are only considering a limited space.

Bias and Variance

Let TD target look *n* steps into the future



TD – high bias, low variance

MC – zero bias, high variance

Policy Iteration

- Policy iteration starts with an arbitrary policy, such as a random policy.
- Then it calculates the value of each state given that policy (*policy evaluation)*.
- Then it updates the policy for every state by calculating the expected reward of each action applicable from that state (*policy improvement*).

Algorithm – Policy Iteration

Input: MDP $M = \langle S, s_0, A, P_a(s' \mid s), r(s, a, s') \rangle$

Output: Policy π

Set V^π to arbitrary value function; e.g., $V^\pi(s) = 0$ for all s.

Set π to arbitrary policy; e.g. $\pi(s) = a$ for all s, where $a \in A$ is an arbitrary action.

Repeat

Compute $V^\pi(s)$ for all s using policy evaluation

For each $s \in S$

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

Until π does not change

Q-Learning

Defining the Q-Function

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots$$

Total reward, R_t, is the discounted sum of all rewards obtained from time t

$$Q(s_t, a_t) = \mathbb{E}[R_t | s_t, a_t]$$

The Q-function captures the **expected total future reward** an agent in state, s, can receive by executing a certain action, a

How to Take Actions Given a Q-Function

$$Q(s_t, a_t) = \mathbb{E}[R_t | s_t, a_t]$$

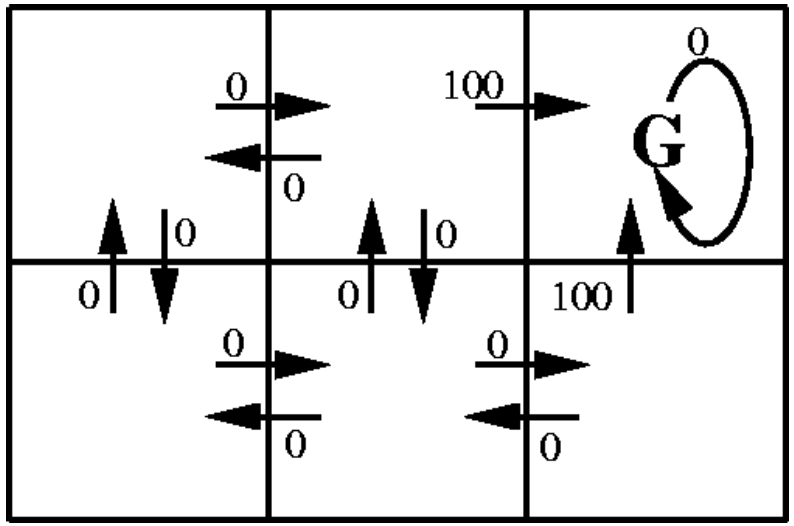
(state, action)

Ultimately, the agent needs a **policy $\pi(s)$**, to infer the **best action to take** at its state, s

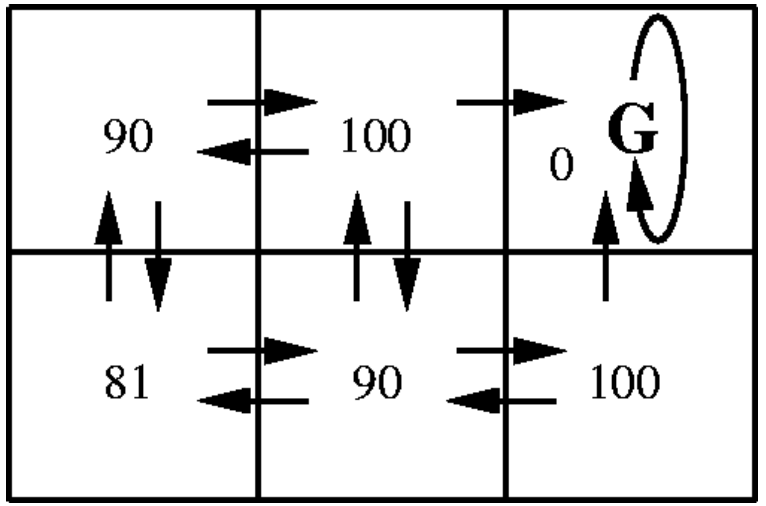
Strategy: the policy should choose an action that maximizes future reward

$$\pi^*(s) = \underset{a}{\operatorname{argmax}}\, Q(s, a)$$

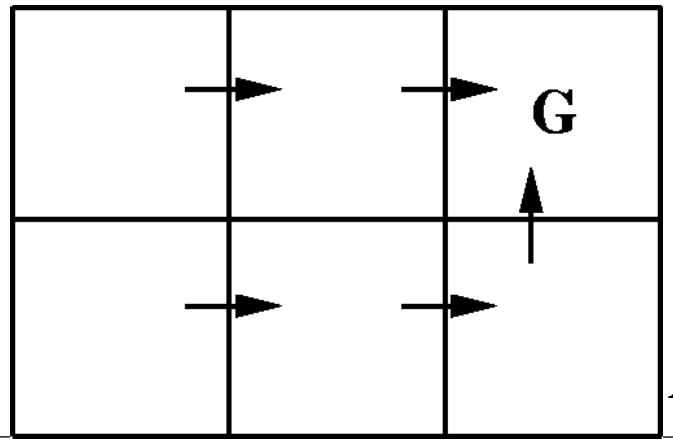
MDP Example



r(s,a)



V(s)*



An optimal policy

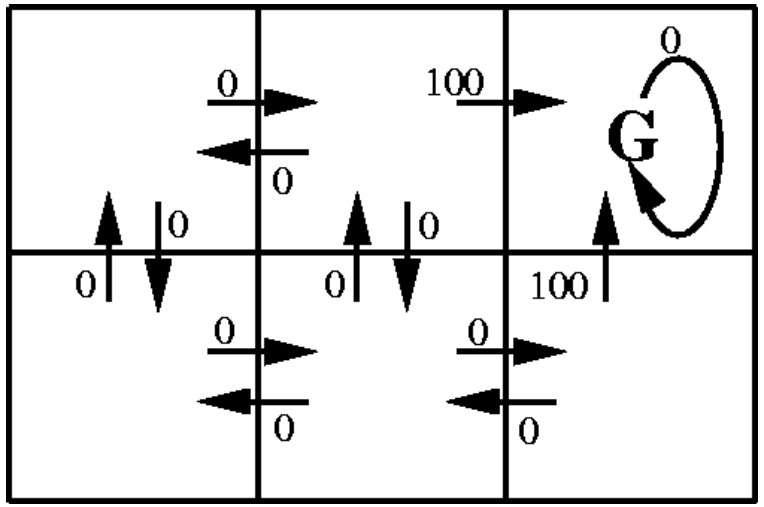
The Q-Function

Optimal policy:

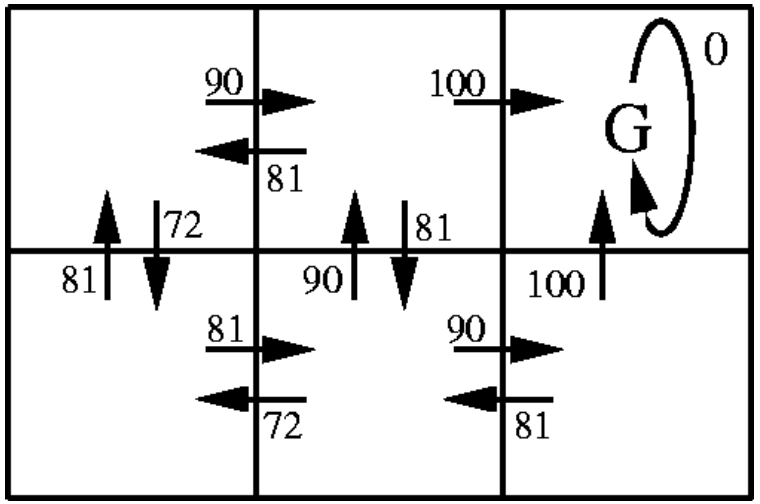
- $\pi^*(s) = \text{argmax}_a[r(s, a) + \gamma V^*(\delta(s, a))]$
- Doesn't work if we don't know r and δ.

The Q-function:

- $Q(s, a) := r(s, a) + \gamma V^*(\delta(s, a))$
- $\pi^*(s) = \text{argmax}_a Q(s, a)$



r(s,a)



Q(s,a)

The Q-Function

- Note Q and V* closely related:
  $V^*(s) = \max_{a'} Q(s, a')$
- Therefore Q can be written as:

  $$Q(s_t, a_t) := r(s_t, a_t) + \gamma V^*(\delta(s_t, a_t)) =$$
  $$r(s_t, a_t) + \gamma \max_{a'} Q(s_{t+1}, a')$$
- If $Q\hat{}$ denote the current approximation of Q then it can be updated by:

  $Q\hat{}(s, a) := r + \gamma \max_{a'} Q\hat{}(s', a')$

Q-Learning for Deterministic Worlds

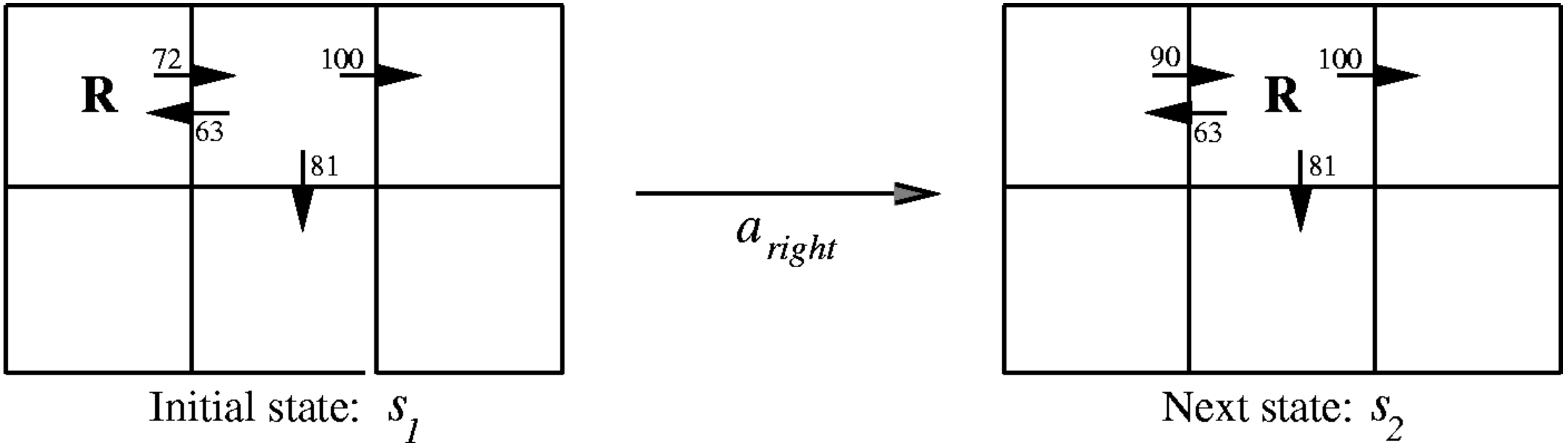
For each s, a initialize table entry $\hat{Q}(s, a) := 0$.

Observe current state s.

Do forever:

1. Select an action a and execute it
2. Receive immediate reward r
3. Observe the new state s'
4. Update the table entry for $\hat{Q}(s, a)$:
   $\hat{Q}(s, a) := r + \gamma \max_{a'} \hat{Q}(s', a')$
5. $s := s'$

Q-Learning Example



$$\hat{Q}(s_1, a_{right}) := r + \gamma \max_{a'} \hat{Q}(s_2, a')$$
$$:= 0 + 0.9 \max\{63, 81, 100\}$$
$$:= 90$$

Q-Learning Continued

- Exploration
  - Selecting the best action
  - Probabilistic choice
- Improving convergence
  - Update sequences
  - Remember old state-action transitions and their immediate reward
- Non-deterministic MDPs

On-Policy vs Off-Policy

- **On-policy** learning
  - Samples its behavior from the current (best) policy function while updating that current policy function.
  - Even when selection explores non-optimally, it follows that to update policy.
  - As it learns the latest policy, it samples from this policy until convergence.
- **Off-policy** learning
  - Samples its behavior from one policy but updates the one with the best rewards.
  - When behavior explores non-optimally, learning exploits; it learns the best policy off the behavior policy, but may not converge, since behavior policy may be not influenced by learning.
  - But it might be a large database of previous samples, and off-policy is suited for parallelization.

Q-Learning vs SARSA

- Use Q to select s' and a', and then:
  - On-policy learning: SARSA

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

  - Off-policy learning: Q-learning

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

SARSA

- Initialize Q-function
- For All Episodes:
  - Initialize s; Select a ϵ-greedy from Q(s)
  - For All Time Steps in this Episode:
    - Perform a in Environment giving s' and r
    - Select a' ϵ-greedy from Q(s) :: **SELECT DOWN**
    - Q(s,a) ← Q(s,a)+α[r+γQ(s',a')-Q(s,a)] :: **LEARN UPDATE**
    - s ← s'; a ← a'
- return Q

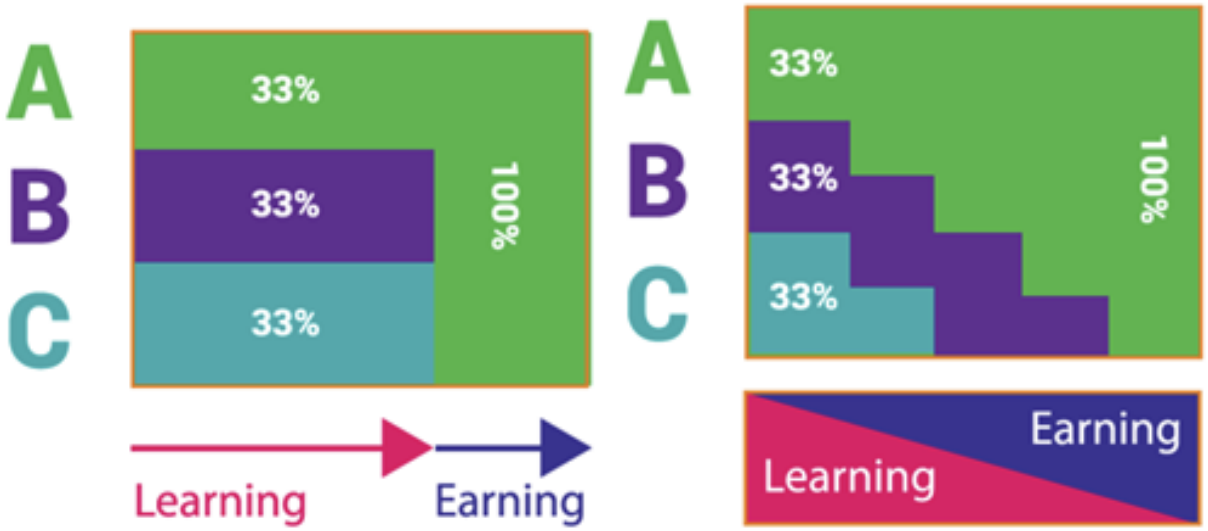
Exploration and Exploitation

Exploration vs Exploitation

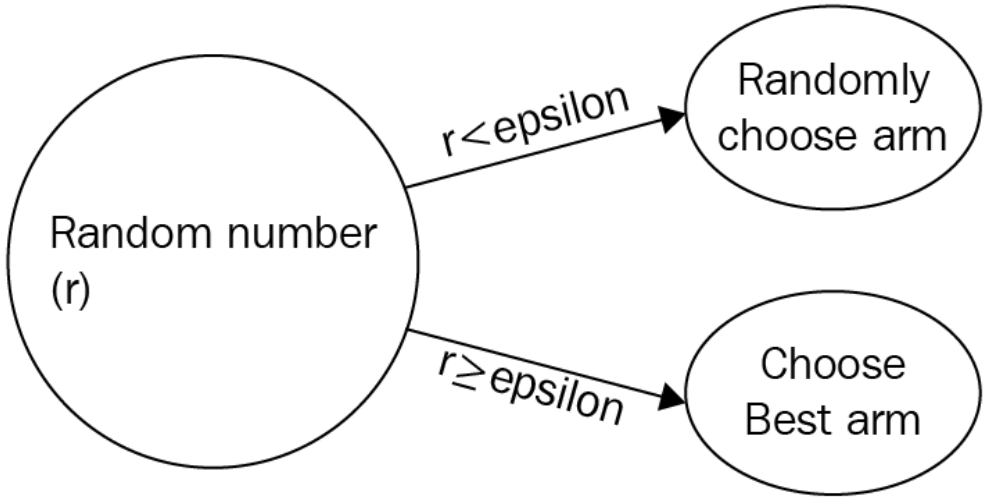
- Exploration corresponds to the idea that an agent must experience as much as possible of its environment to learn how to behave in it.
- Exploitation corresponds to the idea that an agent should maximize its utility based on its current knowledge of the environment.
- The trade-off is between making decisions based on the current knowledge versus learning more about the environment to make better decisions later.
- An optimal action for exploration can be suboptimal for exploitation, and the other way around.
- Pure exploration disallows an agent from using its knowledge, and pure exploitation risks that the agent gets stuck or fails to find the most optimal solution.

Multi-Arm Bandits

- Theory for optimal exploration sampling
- Important in Clinical Trials to find minimal regret



Epsilon-Greedy



- Greedy: Exploit best current action
- However, for an epsilon fraction, you explore a random action
- Static, adaptive schemes

Epsilon-Greedy

- When epsilon-greedy explores [Select Down], and finds that the action was indeed non-optimal, then what [Learn Up]?
- On-policy learning says: use its reward anyway.
  [highly consistent, but perhaps slow convergence]
- Off-policy learning says: use the best action instead to learn from.
  [may diverge, but may be quicker to converge]

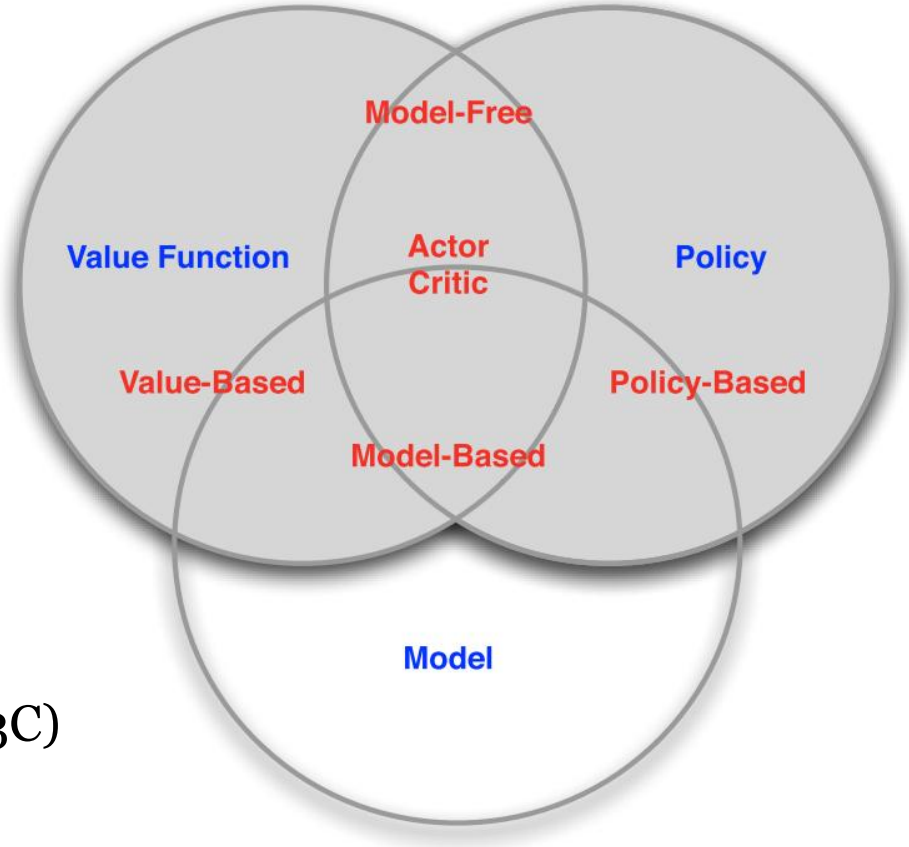
Q-Learning vs SARSA

- Initialize Q-function
- For All Episodes:
  - Initialize s
  - For All Time Steps in this Episode:
    - Select a ϵ-greedy from Q(s) :: **SELECT DOWN**
    - Perform a in Environment giving s' and r
    - $Q(s,a) \leftarrow Q(s,a)+\alpha[r+\gamma \max_a Q(s',a)-Q(s,a)]$ :: **LEARN UPDATE**
    - $s \leftarrow s'$
- return Q

- Initialize Q-function
- For All Episodes:
  - Initialize s; Select a ϵ-greedy from Q(s)
  - For All Time Steps in this Episode:
    - Perform a in Environment giving s' and r
    - Select a' ϵ-greedy from Q(s) :: **SELECT DOWN**
    - $Q(s,a) \leftarrow Q(s,a)+\alpha[r+\gamma Q(s',a')-Q(s,a)]$ :: **LEARN UPDATE**
    - $s \leftarrow s'$; $a \leftarrow a'$
- return Q

Reinforcement Learning Approaches

- Value-Based:
  - Learn value function
  - Implicit policy (e.g. greedy selection)
  - Example: Deep Q Networks (DQN)
- Policy-Based:
  - No value function
  - Learn explicit (stochastic) policy
  - Example: Stochastic Policy Gradients
- Model-Based:
  - Learn transition model
  - Implicit policy
  - Example: Dreamer
- Actor-Critic:
  - Learn value function
  - Learn policy using value function
  - Example: Asynchronous Advantage Actor Critic (A3C)



RL LE1 VT2026:

Positioning Reinforcement Learning in AI
Introduction to Reinforcement learning
Course Overview
Value Iteration and Policy Iteration
Tabular Methods (Q-Learning, SARSA)
Exploration and Exploitation

www.ida.liu.se/~TDDE78

LINKÖPING UNIVERSITY