Sequential Decision Making and Reinforcement Learning

(SDMRL)

Intro to RL

Sarah Keren

The Taub Faculty of Computer Science
Technion - Israel Institute of Technology

Acknowledgments

- David Sliver's course on RL: *https://www.deepmind.com/learning-resources/introduction-to-reinforcement-learning-with-david-silve*
- Slides by Malte Helmert, Carmel Domshlak, Erez Karpas and Alexander Shleyfman.

Do we still need to know about planning?

by Subbarao Kambhampati:
"Human, grant me the serenity to accept the things I cannot learn, the data to learn the things I can, and the wisdom to know the difference."[1]

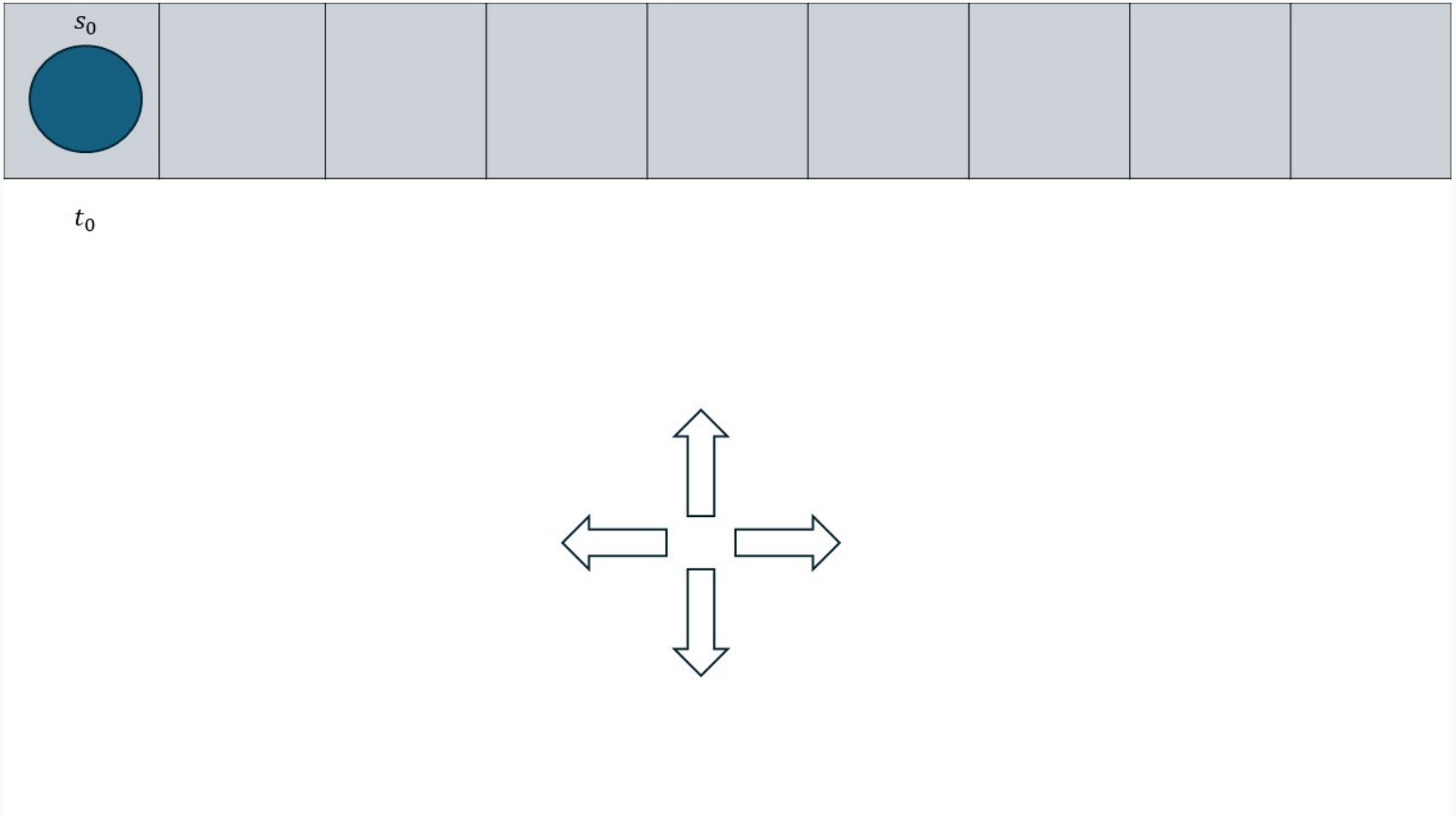
[1] My addition: and the ability to know what to do about it

Intro RL

Example

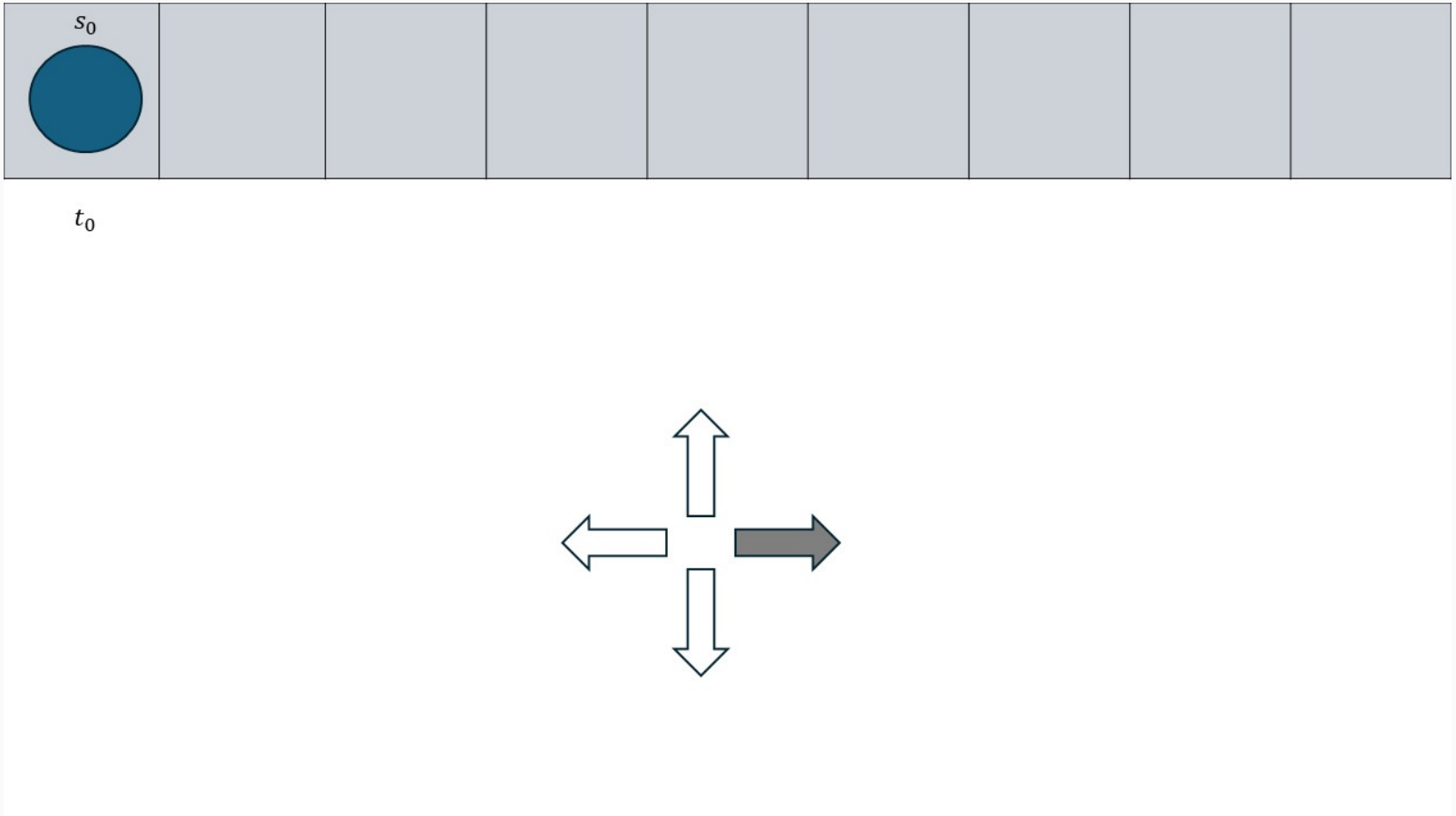
Figure 1: What to do next?

Example



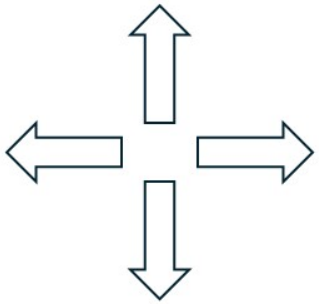
What to do next?

Example



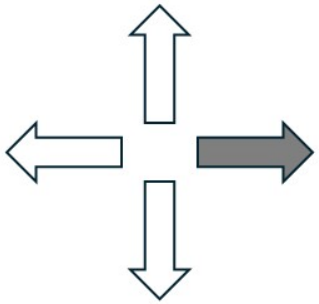
What to do next?

Example



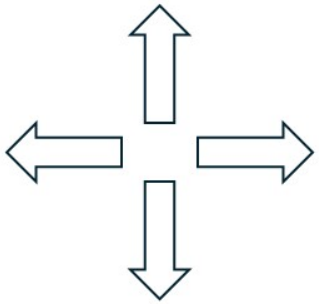
What to do next?

Example



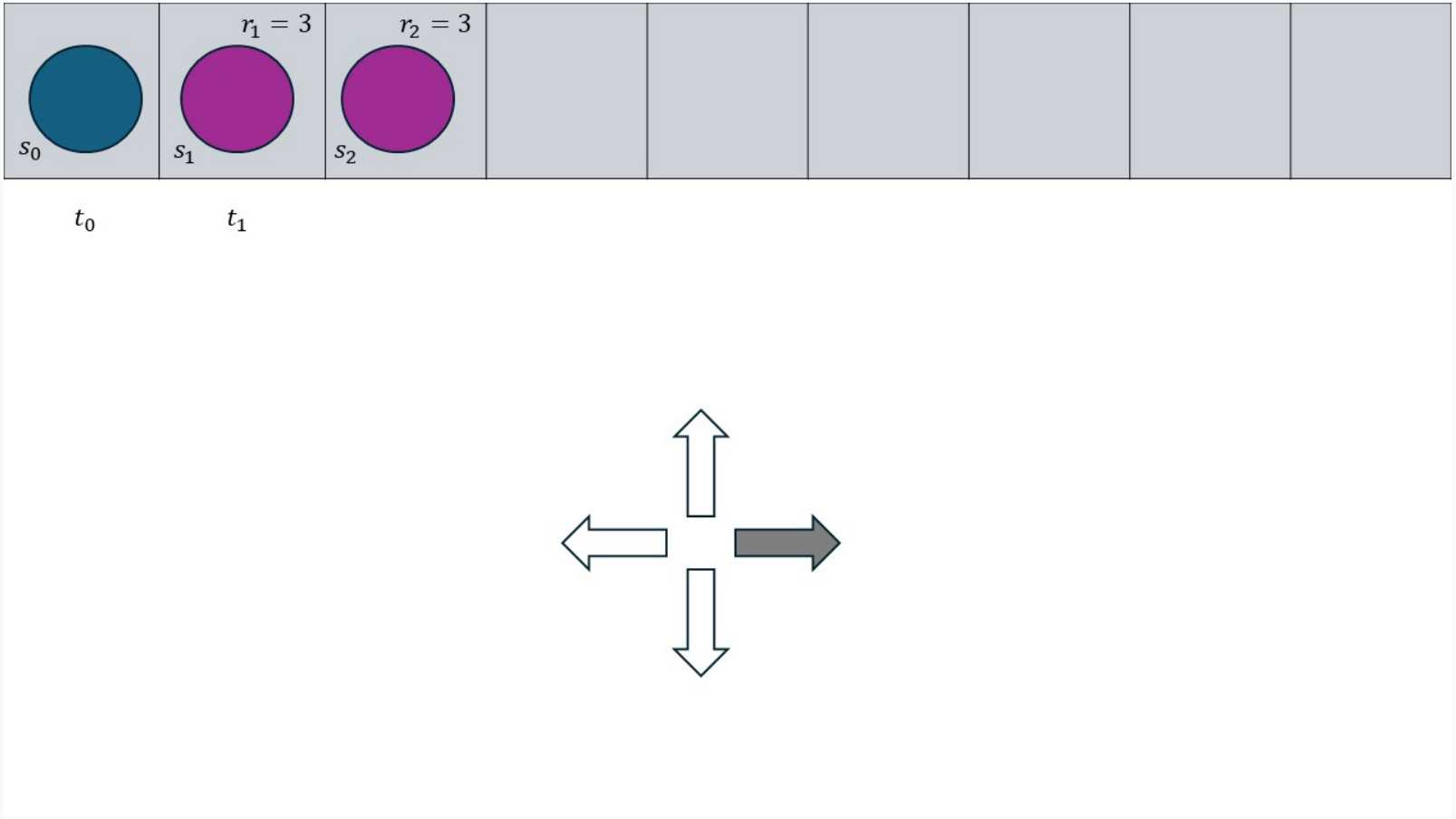
What to do next?

Example



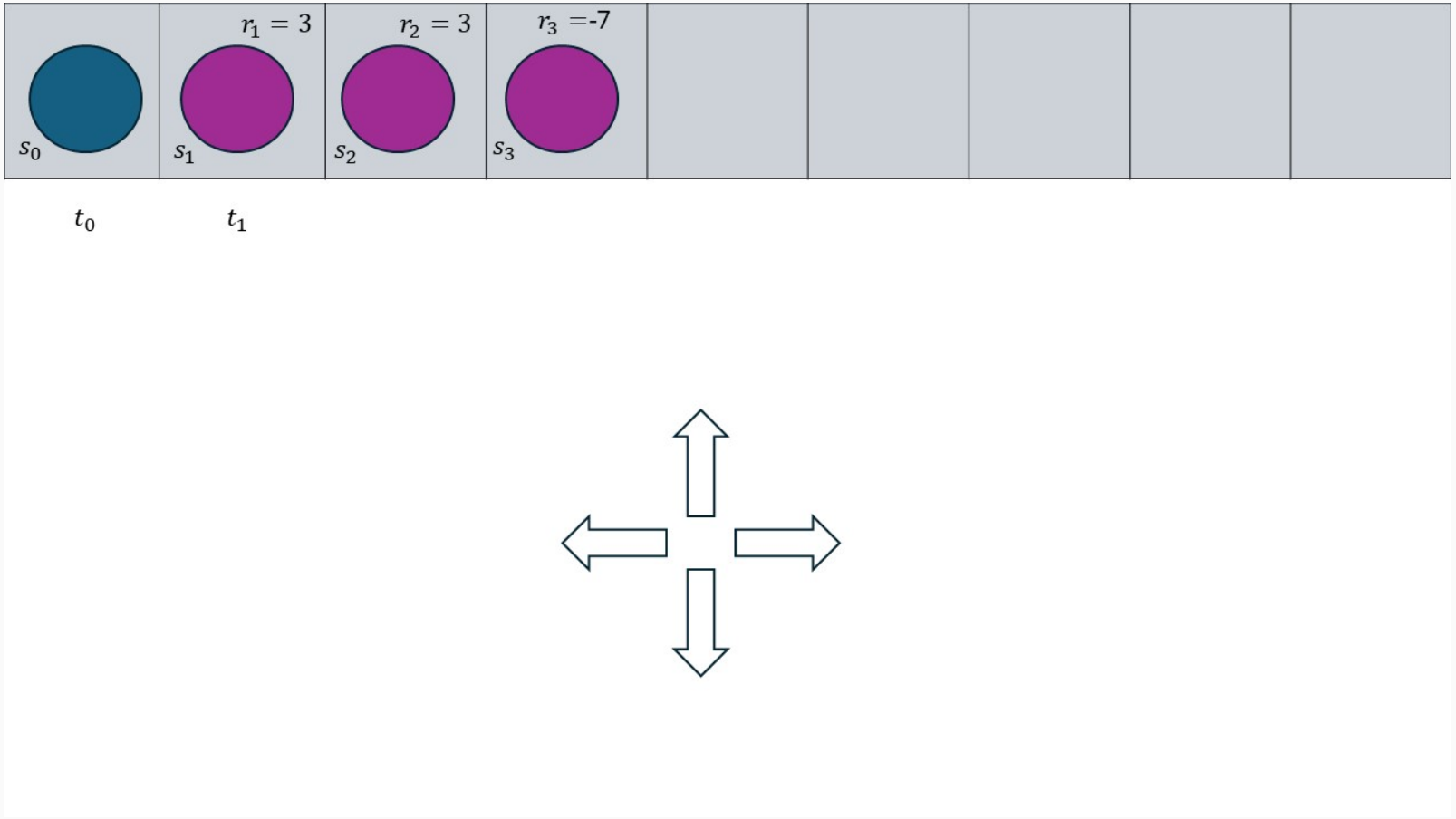
What to do next?

Example



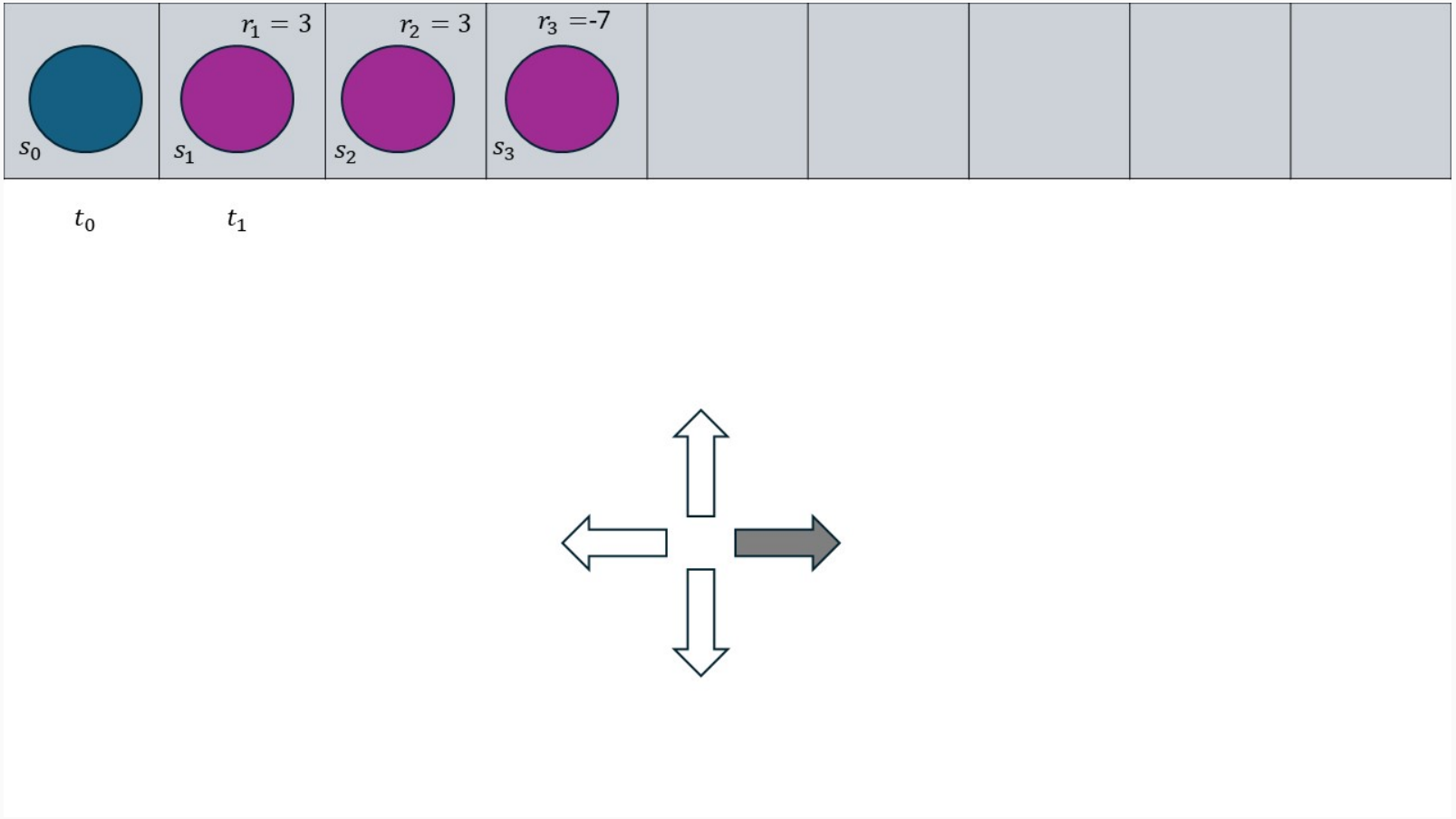
What to do next?

Example



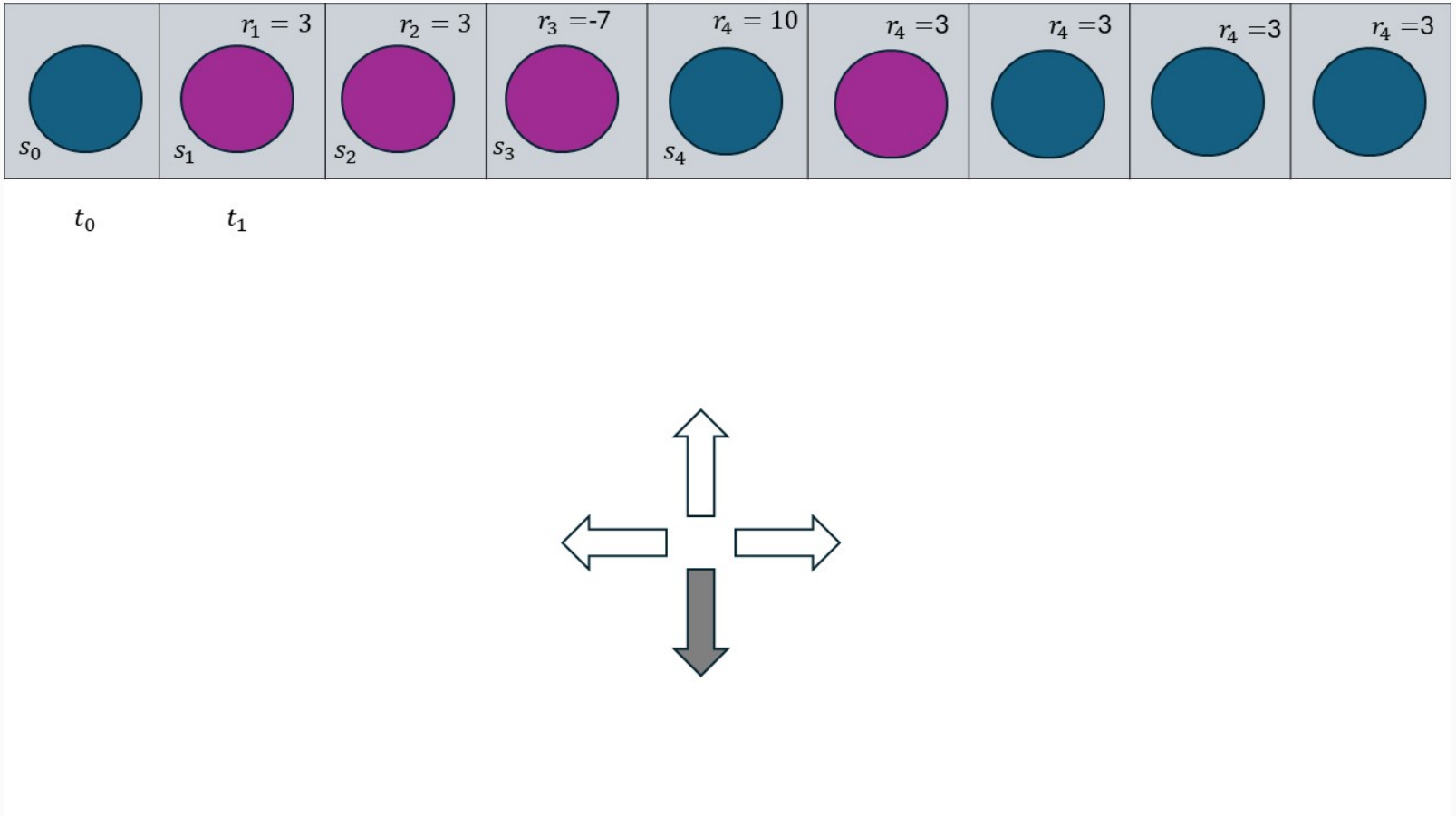
What to do next?

Example



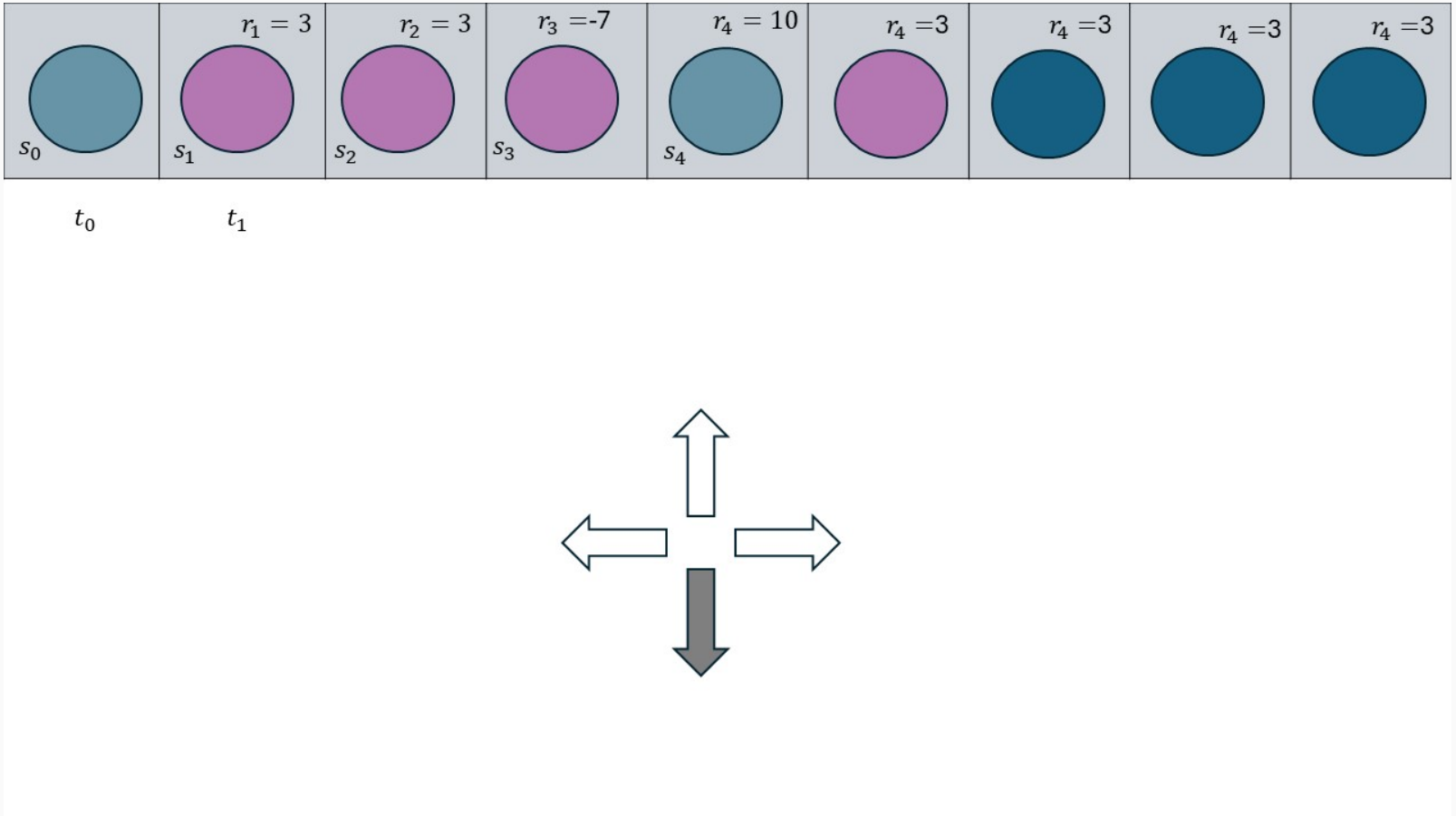
What to do next?

Example



What to do next?

Example

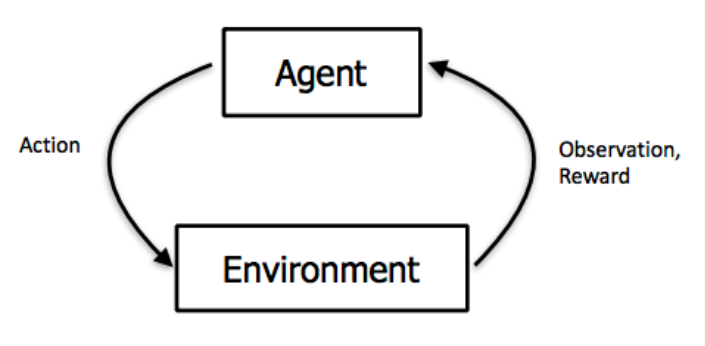


What to do next?

What is RL ? - Sutton and Barto 2018

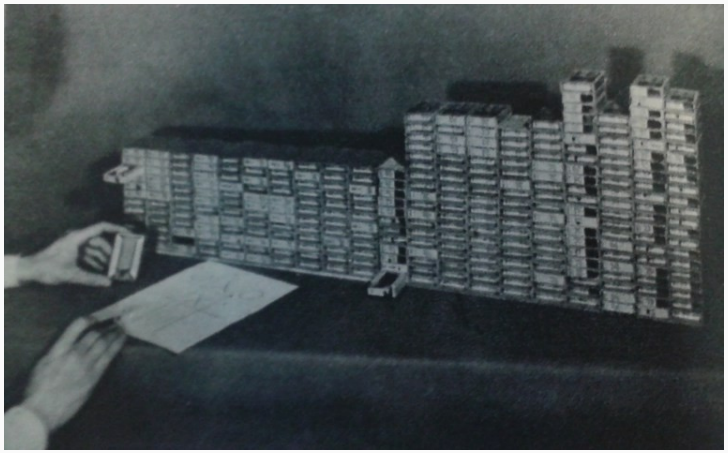
Reinforcement learning is learning what to do - how to **map situations to action** - so as to maximize a numerical reward signal.

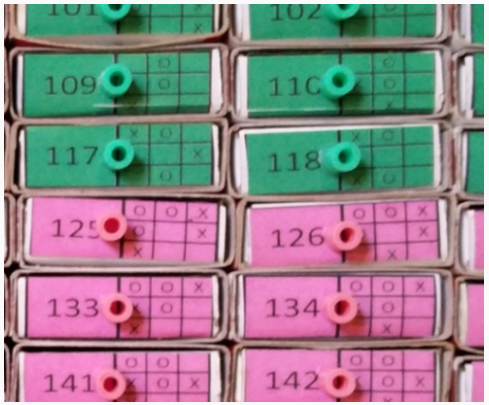
" Reinforcement learning is a computational approach to understanding and automating goal-directed learning and decision-making. It is distinguished from other computational approaches by its emphasis on learning by an agent from **direct interaction** with its environment, without relying on exemplary supervision or complete models of the environment.



Match Box RL

In 1960 Donald Michie designed MENACE a large pile of matchboxes that contained a number of beads and learned to play tic-tac-toe.





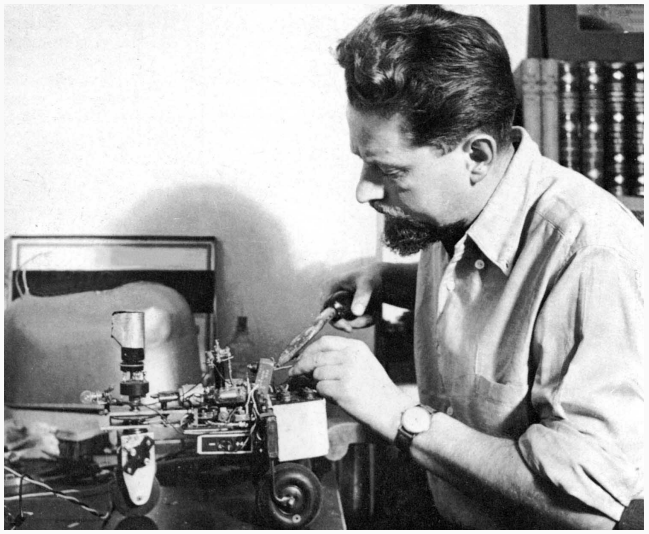
MENACE works a little like a neural network. It is randomly initialized, but after playing a few games it adjusts itself to favour moves that are supposedly more successful in each situation. Depending on the model's success, it will either be punished or rewarded.

When training begins, all boxes contain colour-coded beads, where each colour represents a move (or position) on a board.

At the end of each game, if MENACE loses, each bead MENACE used is removed from each box. If MENACE wins, three beads the same as the colour used during each individual turn are added to their respective box. If if the game resulted in a draw, one bead is added.

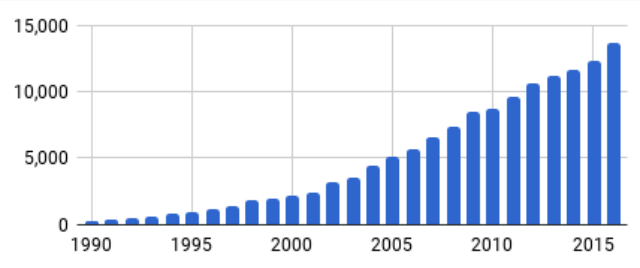
First (?) RL Robots

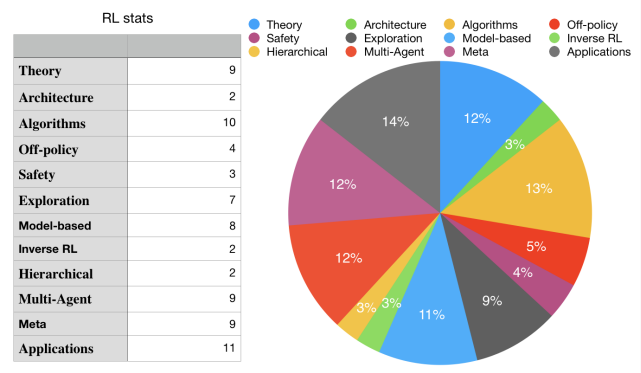
Grey Walter (1910-1977; 1940s) Neurophysiologist and Cyberneticist. Created simple robots based on reflexes (before AI existed).

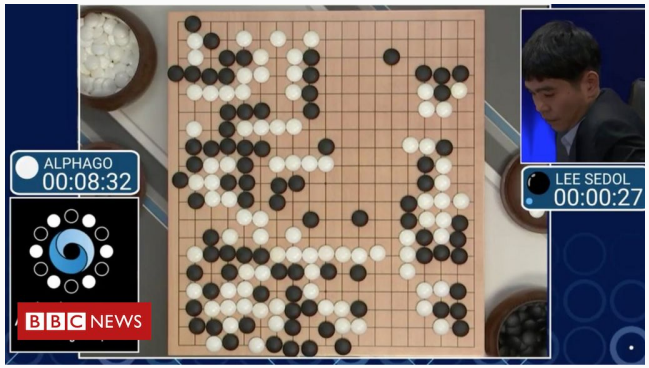


https://www.youtube.com/watch?v=lLULRlmXkKo&t=1s

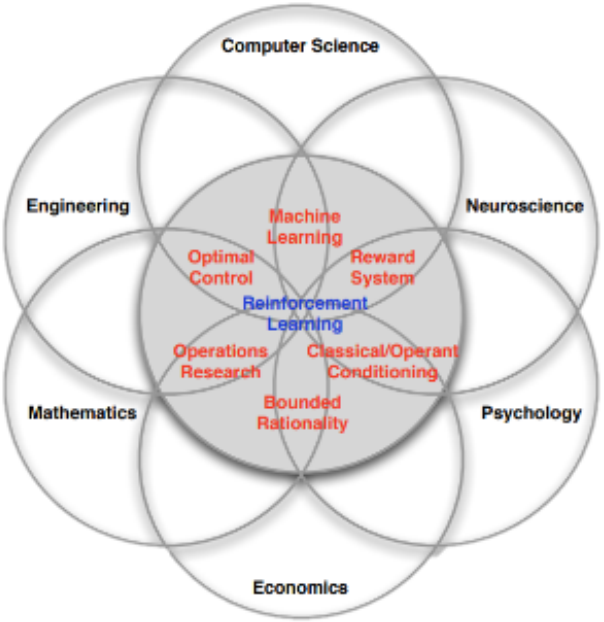
RL in recent years





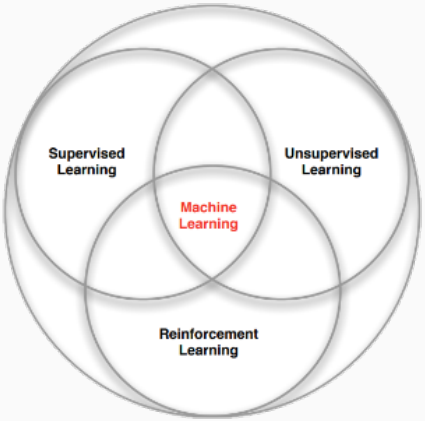


What is RL



By David Silver

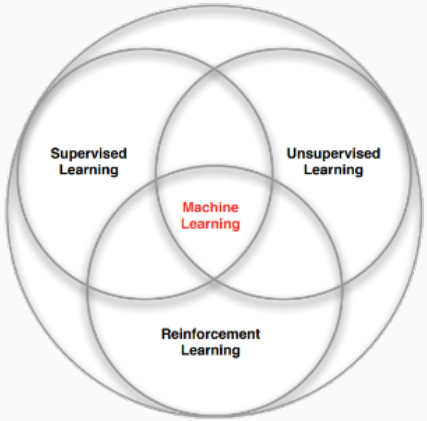
RL as a special case of ML



What makes reinforcement learning different from other machine learning paradigms?

By David Silver

RL as a special case of ML



What makes reinforcement learning different from other machine learning paradigms?

- There is no supervisor, only a reward signal
- Feedback is delayed, not instantaneous
- Time really matters (sequential, non i.i.d data)
- Agent's actions affect the subsequent data it receives

By David Silver

RL as a generalization of model-based planning

Two fundamental problems in sequential decision making

Model-Based Planning:

- A model of the environment is known.
- The agent performs computations with its model (without any external interaction)
- The agent improves its policy a.k.a. deliberation, reasoning, introspection, pondering, thought, search

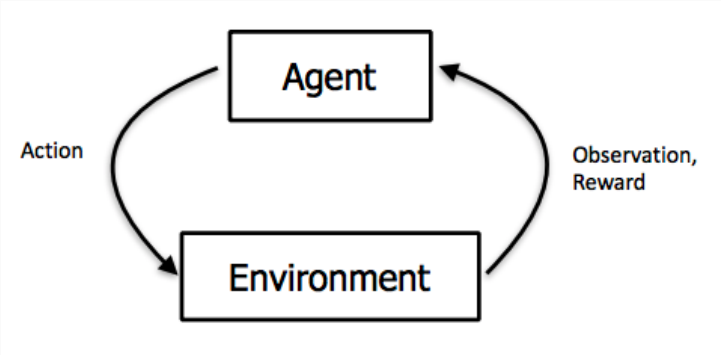
Reinforcement Learning:

- The environment is initially unknown.
- The agent interacts with the environment and improves its policy.

Basics of RL

Characteristics of an RL Problem

- RL is **closed-loop**:
  - don't have direct instructions as to what actions to take
  - consequences of actions, including reward signals, play out over extended time periods



The environment is assumed to be modeled as an MPD or POMDP : even when the model is not known or explicitly learned!

Reward

- At each step the agent receives a reward signal
- This is the way to induce desired behaviors
- Very challenging

https://www.youtube.com/watch?v=xHEMkbyXFxs&t=136s
https://www.youtube.com/watch?v=GFiWEjCedzY

How do we learn in RL ? What is the structure of the data ?

Trajectory/Rollout/Episode

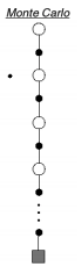
- In Reinforcement Learning (RL), a trajectory (or episode or rollout), is a sequence of states, actions, and rewards that an agent experiences as it interacts with the environment from a starting state until a terminal state is reached.
- Formally, a trajectory can be defined as:

$$\tau = s_0, a_1, r_1, s_1, \ldots, s_{T-1}, a_{T-1}, r_T, s_T$$

where:

- s_i represents the state of the environment at time step i.
- a_i represents the action taken by the agent at time step i.
- r_{i+1} represents the reward received by the agent after taking action a_i in state s_i
- T is the time step at which the episode ends, which can either be a fixed time horizon or when a terminal state is reached.

Trajectory/Rollout/Episode



Policy

- A **deterministic** policy is a mapping:
  - $\pi : \mathcal{S} \rightarrow \mathcal{A}$ from states to actions
  - $\pi : \mathcal{B} \rightarrow \mathcal{A}$ from beliefs to action
  - $\pi(s)/\ \pi(\beta)$ is the (single) action the agent will perform at state s or belief β.
- A **non-deterministic** policy is a mapping:
  - $\pi : \mathcal{S} \rightarrow \mathcal{A}^n$ from states to actions,
  - $\pi : \mathcal{B} \rightarrow \mathcal{A}^n$ from beliefs to actions
  - $\pi(s)/\ \pi(\beta)$ is the set of actions one of which the agent will perform at state s or belief β.
- A **stochastic** policy is a mapping:
  - $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ from states to actions
  - $\pi : \mathcal{B} \times \mathcal{A} \rightarrow [0, 1]$ from beliefs to action
  - $\pi(s, a)/\ \pi(\beta, a)$ is the probability the agent will perform action a at state $s/$ belief β.

If we know there is an optimal **deterministic** policy for any MDP, why do we need a **probabilistic** policy?

Policy Parameters

- Instead of using a policy $\pi(a|s)$ that explicitly maps states (or beliefs to actions), we can use a **parameterized policy** $\pi_\theta(a|s)$, i.e., as a function with parameters θ.
- For deterministic policies θ specify the action taken at each state, and for stochastic policies, they induce a probability distribution over actions.
- Examples:
  - for linear policies $\pi_\theta(s) = \theta^\intercal s$ where θ is a vector of weights.
  - for neural network policies θ are the weights of the neural network.

Reminder: Return and Value

- Return is (typically) the total discounted reward:

$$G_t = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{T-1} R_T$$

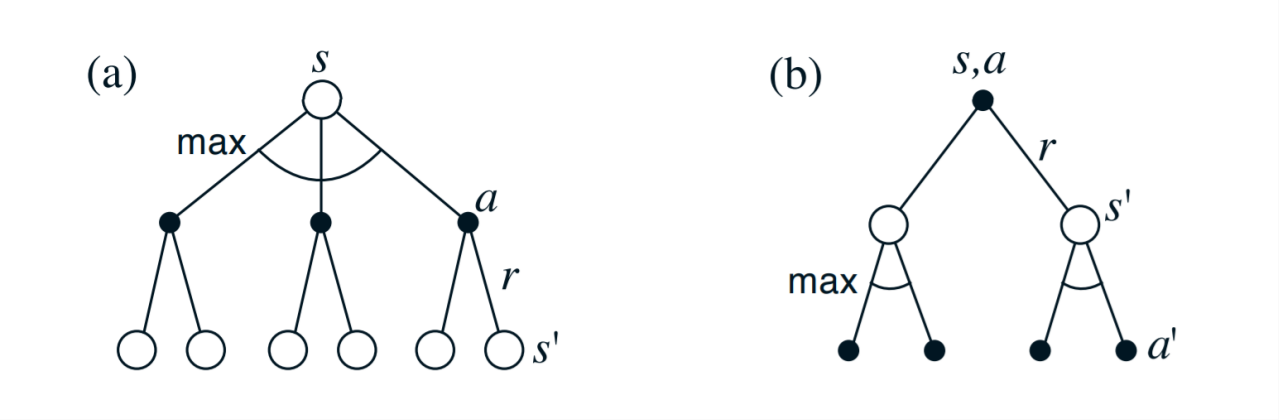
- Value function is (typically) the expected return for following the policy π from state s:

$$\mathcal{V}_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$$

- Q-value function is the expected return for following the policy π from state s after taking action a:

$$Q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$$

Value Functions



Backup diagrams for $\mathcal{V}_\pi(s)$ (left) and $Q_\pi(s,a)$ (right).

Since it may not be practical to keep separate averages for each state individually $\mathcal{V}_\pi$ and Q_π are often represented as parameterized functions (with fewer parameters than states).

Terminology

- Bootstrapping
- Sampling
- Stochastic Approximation
- Importance Sampling

Terminology

- **Bootstrapping** - estimating a value based on another estimation. For example, we can estimate the value of state based on the value of its consecutive states.
- Sampling
- Stochastic Approximation
- Importance Sampling

Terminology

- **Bootstrapping** - estimating a value based on another estimation. For example, we can estimate the value of state based on the value of its consecutive states.
- **Sampling** - selecting a subset of individuals from within a statistical population to estimate characteristics of the whole population.
- **Stochastic Approximation**

- **Importance Sampling**

Terminology

- **Bootstrapping** - estimating a value based on another estimation. For example, we can estimate the value of state based on the value of its consecutive states.
- **Sampling** - selecting a subset of individuals from within a statistical population to estimate characteristics of the whole population.
- **Stochastic Approximation** - using random samples of a function to efficiently approximate its properties (e.g., use samples of trajectories to approximate the expected value of following a policy from a given state).
- **Importance Sampling**

Terminology

- **Bootstrapping** - estimating a value based on another estimation. For example, we can estimate the value of state based on the value of its consecutive states.
- **Sampling** - selecting a subset of individuals from within a statistical population to estimate characteristics of the whole population.
- **Stochastic Approximation** - using random samples of a function to efficiently approximate its properties (e.g., use samples of trajectories to approximate the expected value of following a policy from a given state).
- **Importance Sampling** - estimating expected values under one distribution given samples from another. **When would this be relevant?**
  - Estimations are based on the importance-sampling ratio, i.e., the relative probability of an occurrence (e.g., trajectory) occurring under the target and behavior policy.

How to find an optimal policy ?

RL Solution Approaches

Reinforcement Learning (RL)

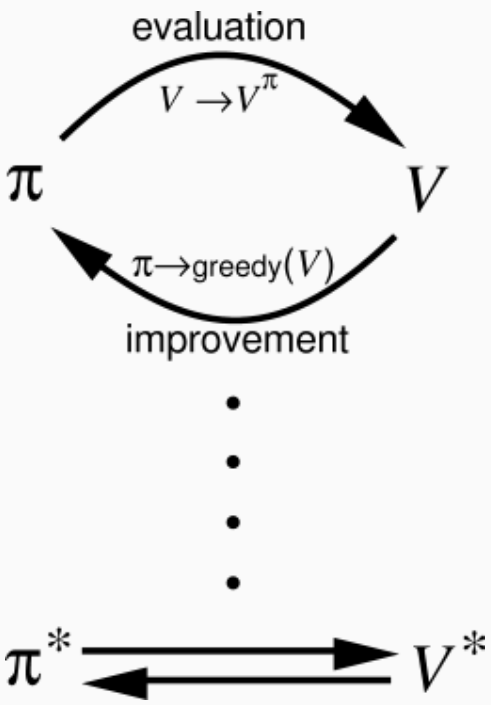
- When we used planning, we assumed a complete model of the world was given.
  Did this mean we assumed the state of the world is always observable ?
- With RL we want to account for settings with no/ partial knowledge of environment.
- Can act in the world and observe states and reward.
- We will assume the world behaves as an MDP.
  Is this overly restrictive ?

A note on scope

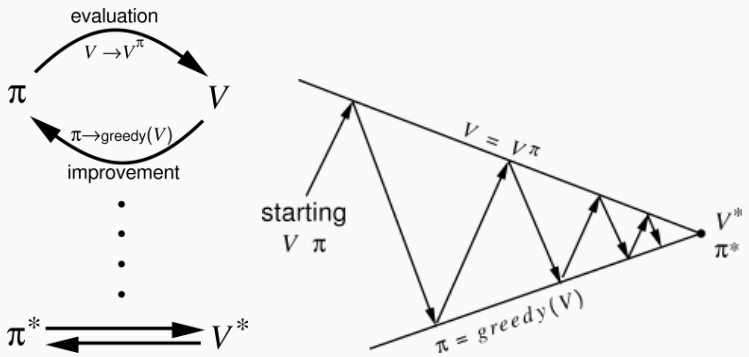
- Most methods we consider are structured around estimating value functions, but other approaches exist.
- For example, methods such as genetic algorithms, genetic programming, simulated annealing, and other optimization methods have been used to approach reinforcement learning problems without ever appealing to value functions.

Anatomy of RL Algorithms

- **Prediction / Evaluation:** evaluate the future given a policy
- **Control/ Update / Improvement:** optimize the policy.



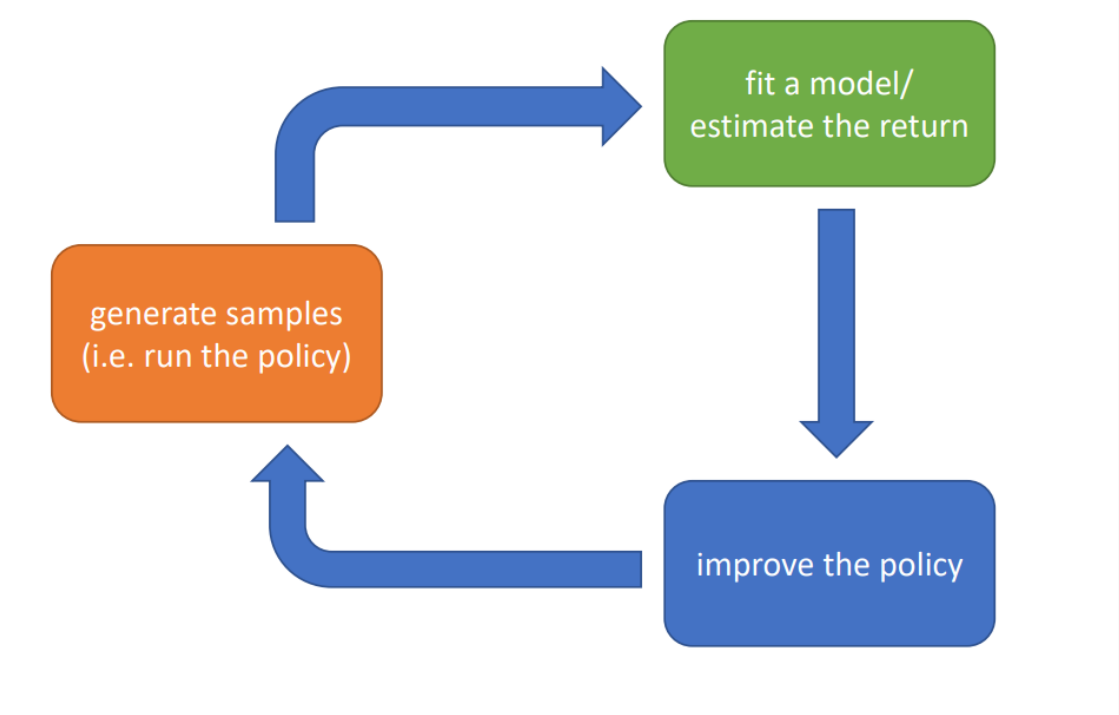
Anatomy of RL Algorithms



Policy prediction (evaluation) vs. Control

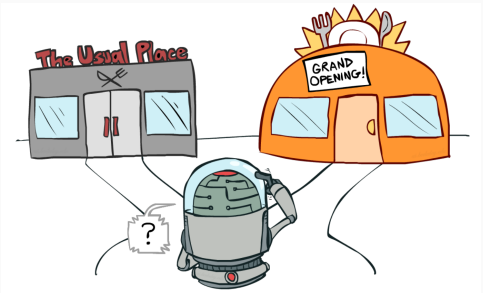
- **Prediction (Estimation)** - assessing the expected value of following a given policy.
  - Often serves as a component of active learning algorithms
- **Control**: Improving a policy / optimising the value function of an unknown MDP
  - Analogous to solving the underlying MDP, but without first being given the MDP model.

Anatomy of RL Algorithms (Sergey Levine's view)



Exploration vs. Exploitation Tradeoff

- In reinforcement learning there's a tradeoff between **exploration** (trying new actions e.g., choosing a random action) and **exploitation** (choosing actions based on already learned values).
- To obtain a lot of reward, an agent must prefer actions that it has tried in the past. But to discover such actions, it needs to try actions it has not selected before.



Ideas for managing the trade-off ?

Exploration-Exploitation Tradeoff

- Managing the trade-off between exploration and exploitation is a critical issue in RL
- Researched for many decades, yet remains unresolved.
- Does not arise in supervised and unsupervised learning. Why ?
- Basic intuition behind most approaches:
  Explore more when knowledge is weak
  Exploit more as we gain knowledge

Exploration-Exploitation Tradeoff: Example Approaches

- **Epsilon-Greedy Policy:** one of the simplest methods to balance exploration and exploitation:
  - With probability ϵ, a random action is chosen (exploration),
  - and with probability $1 - \epsilon$, the best-known action is chosen (exploitation).
- Other approaches (which we will explore later on):
  - softmax policy / Boltzmann exploration
  - Upper Confidence Bound (UCB)

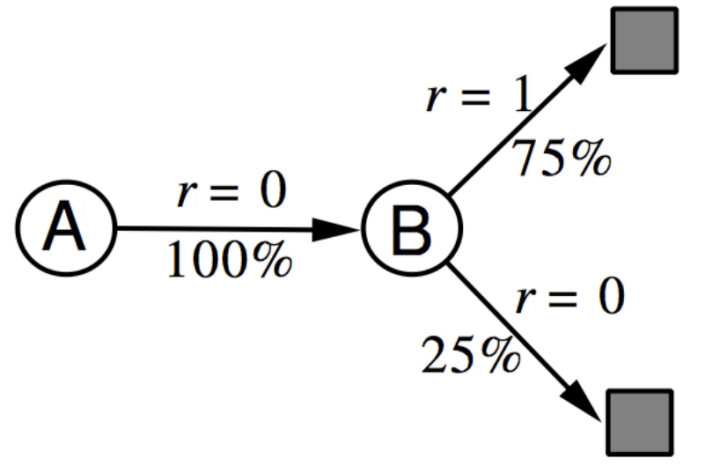
Reminder: Environment Model

- The environment is assumed to be an MDP or POMDP.
- This is the case even if we choose not to learn the model.

AB Example

Two states A, B; no discounting; 8 episodes of experience

- A, 0, B, 0
- B, 1
- B, 1
- B, 1
- B, 1
- B, 1
- B, 1
- B, 0



What is $V(A)$, $V(B)$?

Update Rules

$$\mathcal{V}^*(s)$$

Update Rules

$$\mathcal{V}^*(s)$$

max Q over actions:

$$= \max_{a \in A(s)} Q_{\pi^*}(s, a)$$

Update Rules

$$\mathcal{V}^*(s)$$

max Q over actions:

$$= \max_{a \in A(s)} Q_{\pi^*}(s, a)$$

expected return from step t onward:

$$= \max_a \mathbb{E}_{\pi^*}\left[G_t | S_t = s, A_t = a\right]$$

Update Rules

$$\mathcal{V}^*(s)$$

max Q over actions:

$$= \max_{a \in A(s)} Q_{\pi^*}(s, a)$$

expected return from step t onward:

$$= \max_a \mathbb{E}_{\pi^*} [G_t | S_t = s, A_t = a]$$

expected return from step t onward (k counts future steps):

$$= \max_a \mathbb{E}_{\pi^*} \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a \right]$$

Update Rules

the maximal expected return for taking a at s at time t , receiving R_{t+1} and thereafter following an optimal:

$$= \max_a \mathbb{E}\left[R_{t+1} + \gamma \mathcal{V}^*(S_{t+1}) | S_t = s, A_t = a\right]$$

Update Rules

the maximal expected return for taking a at s at time t , receiving R_{t+1} and thereafter following an optimal:

$$= \max_a \mathbb{E}\left[R_{t+1} + \gamma \mathcal{V}^*(S_{t+1}) | S_t = s, A_t = a\right]$$

the maximal expectation over next state s' return r when taking a at s, computed as the immediate return r plus the discounted optimal value of s':

$$= \max_{a \in A(s)} \sum_{s',r} \mathcal{P}(s', r | s, a) \left[r + \gamma \mathcal{V}^*(s')\right]$$

Bellman optimality (considering state transition uncertainty and reward uncertainty separately):

$$\mathcal{V}^*(s) = \max_a \sum_{s'} \mathcal{P}(s' | s, a) \left[\mathcal{R}(s, a, s') + \gamma \mathcal{V}^*(s')\right]$$

Update Rules

the maximal expected return for taking a at s at time t , receiving R_{t+1} and thereafter following an optimal:

$$= \max_a \mathbb{E}\left[R_{t+1} + \gamma \mathcal{V}^*(S_{t+1}) | S_t = s, A_t = a\right]$$

the maximal expectation over next state s' return r when taking a at s, computed as the immediate return r plus the discounted optimal value of s':

$$= \max_{a \in A(s)} \sum_{s',r} \mathcal{P}(s', r|s, a) \left[r + \gamma \mathcal{V}^*(s')\right]$$

Bellman optimality (considering state transition uncertainty and reward uncertainty separately):

$$\mathcal{V}^*(s) = \max_a \sum_{s'} \mathcal{P}(s'|s, a) \left[\mathcal{R}(s, a, s') + \gamma \mathcal{V}^*(s')\right]$$

similar development for q functions:

$$Q^*(s, a) = \sum_{s',r} \mathcal{P}(s', r|s, a) \left[r + \gamma \max_{a'} Q^*(s', a')\right]$$

$$Q^*(s, a) = \sum_{s'} \mathcal{P}(s'|s, a) \left[\mathcal{R}(s, a, s') + \gamma \max_{a'} Q^*(s', a')\right]$$

How do we switch from optimality equations to update formulas?

$$\mathcal{V}^*(s) = \max_a \mathbb{E}_{\pi^*}\left[G_t | S_t = s, A_t = a\right]$$

How can the optimality equation be used within a solution algorithm ?

How do we switch from optimality equations to update formulas?

$$\mathcal{V}^*(s) = \max_a \mathbb{E}_{\pi^*}\left[G_t | S_t = s, A_t = a\right]$$

How can the optimality equation be used within a solution algorithm ?

$$\mathcal{V}_{k+1}(s) = \max_a \mathbb{E}\left[G_t | S_t = s, A_t = a\right]$$

$$\mathcal{V}(s) \leftarrow \mathcal{V}(s) + \alpha(G - \mathcal{V}(s)$$

How do we switch from optimality equations to update formulas?

$$\mathcal{V}^*(s) = \max_a \sum_{s'} \mathcal{P}(s'|s,a)\left[\mathcal{R}(s,a,s') + \gamma \mathcal{V}^*(s')\right] \tag{1}$$

How can the Bellman optimality equation be used within a solution algorithm ?

How do we switch from optimality equations to update formulas?

$$\mathcal{V}^*(s) = \max_a \sum_{s'} \mathcal{P}(s'|s,a)\left[\mathcal{R}(s,a,s') + \gamma \mathcal{V}^*(s')\right] \tag{1}$$

How can the Bellman optimality equation be used within a solution algorithm ?

$$\mathcal{V}_{k+1}(s) = \max_a \sum_{s'} \mathcal{P}(s'|s,a)\left[\mathcal{R}(s,a,s') + \gamma \mathcal{V}_k(s')\right]$$

$$\mathcal{V}(s) \leftarrow \mathcal{V}(s) + \alpha(r + \gamma \mathcal{V}(s') - \mathcal{V}(s))$$

Model-Free vs. Model-Based RL

Model-based approach to RL

- learn the MDP model, or an approximation of it
- use it for policy evaluation or to find an optimal policy

Model-free approach to RL

- derive an optimal policy without explicitly learning the model.
- useful when model is difficult to represent and/or learn

We will consider both types of approaches

Model-Free vs. Model-Based RL

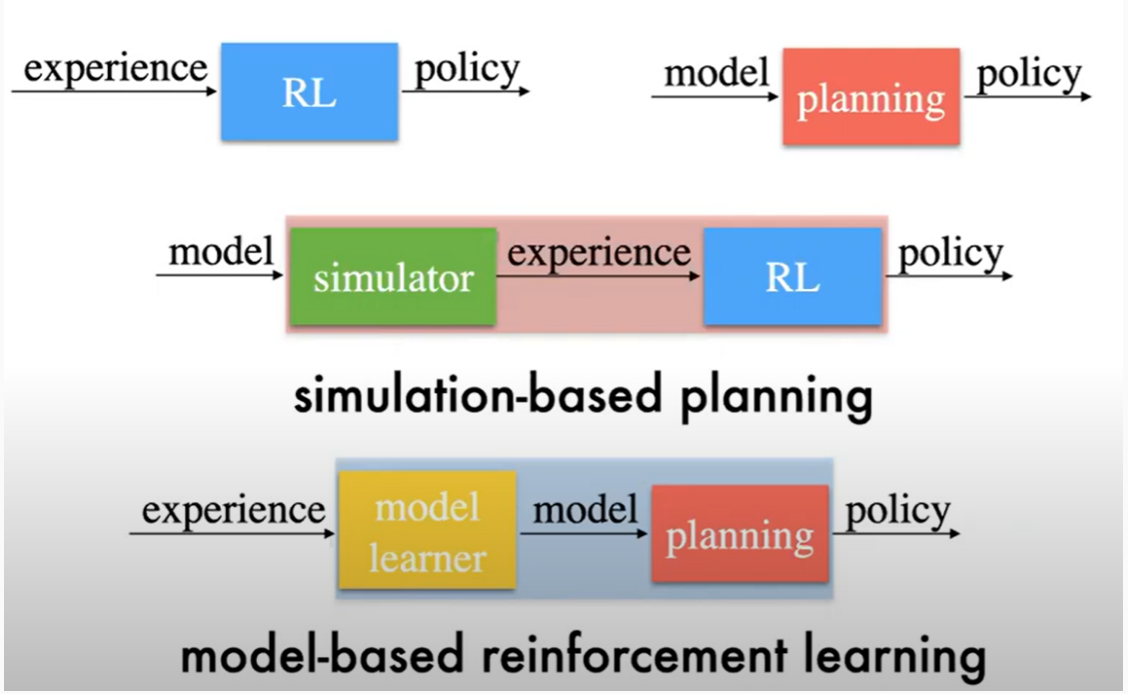


Figure 2: By Michael Littman

https://www.youtube.com/watch?v=45FKxa3qPHo

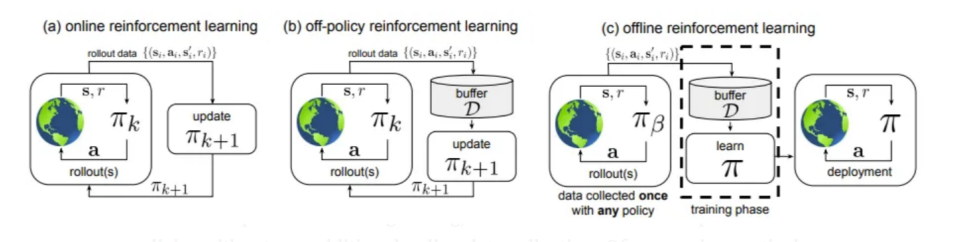
On-Policy vs. Off-Policy

On-Policy Learning

- "Learn on the job"
- Learn about policy π from experience sampled from π

Off-Policy Learning

- "Look over someone's shoulder"
- Learn about policy π from experience sampled from μ

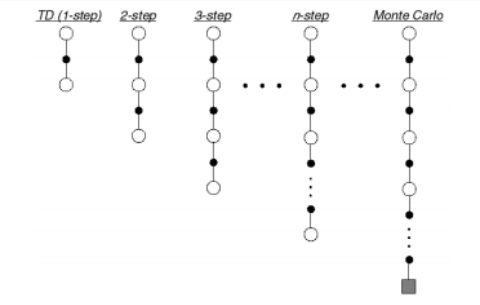


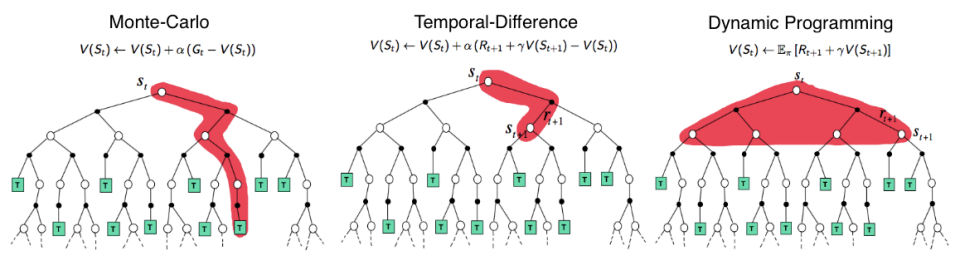
Monte Carlo vs. TD methods

- **Monte Carlo Methods:**
  - In general, Monte Carlo describes randomized algorithms.
  - For RL, they sample fully trajectories in the environment.
  - Estimate the value of a state based on the average the returns we observe after visiting a state.
  - As its interacts with the environment and observe more returns, the average should converge to the expected value.
  - Relevant only to episodic tasks.
- **Temporal Difference (TD) Methods:**
  - Combine Monte-Carlo with dynamic programming (DP).
    - Like Monte-Carlo they learn directly from raw experience by interacting with the environment without a model of its dynmaics
    - Like DP, they update estimates based in part on other learned estimates without waiting for a final outcome (they **bootstrap**).

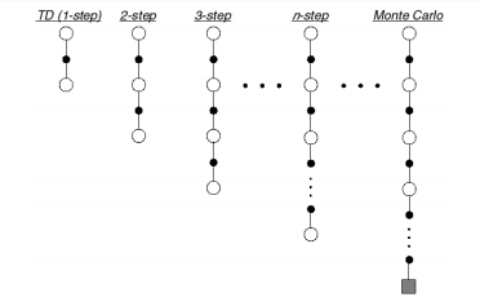
https://towardsdatascience.com/introduction-to-reinforcement-learning-rl-part-5-monte-car

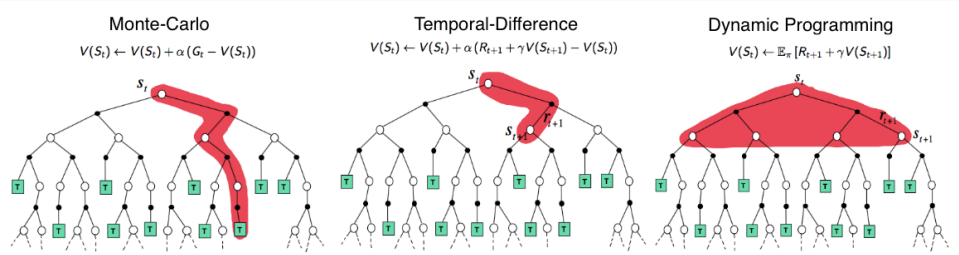
Monte Carlo vs. TD methods

- 



Monte Carlo vs. TD methods





Both approaches are model-free:

- learn from *experiences*:
  - sampled sequences of states, actions and rewards (from actual or simulated interaction with the environment).
- Model of environment only needed for generating simulated experience.

Four Reinforcement Learning Solution Approaches

- **Value-based:** estimate value function or Q-function of the optimal policy (e.g., q-learning)
- **Policy gradients:** Directly learn a parameterized policy without consulting a value function (e.g. Monte-carlo Policy Gradient).
- **Actor-Critic:** combining policy gradient and value-based approaches: The "Critic" estimates the value function (q or v). The "Actor" updates the policy distribution in the direction suggested by the Critic.
- **Model-based RL:** estimate the transition model, and then use it for planning with or without an explicit policy.

Deep learning versions of all approaches.

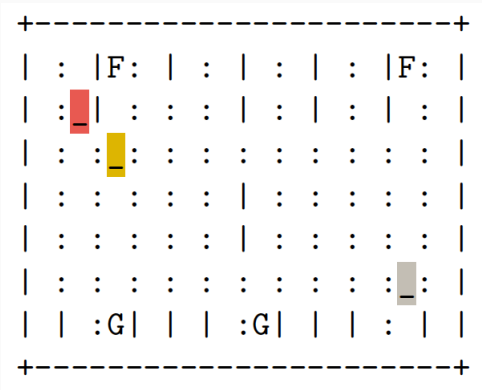
Characterizing an RL Algorithm

- What is the assumed model of the environment ?
- How are policy evaluation and policy update implemented ?
- Value or policy based ?
- Off-policy or on-policy ?
- Online or offline ?
- Model-based or model-free?
- For model-free: Monte-Carlo or Temporal Difference (TD) ?

How to come up with 'good' policies?
If we want to avoid collision?
If we want to turn left at the next intersection?
If we want to buy apples and make it on time for dinner but have fuel for about 30 mins?

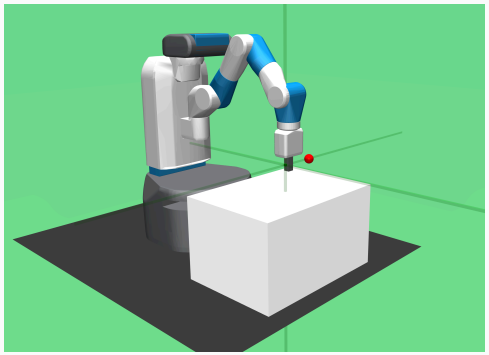
RL for these domains ?

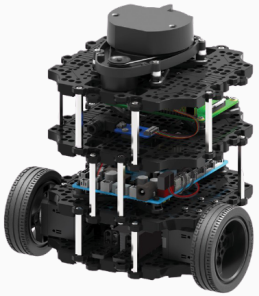
What would be a good solution approach ?



Robotic domains

What would be a good solution approach ?





Many factors make RL difficult

- Actions have non-deterministic effects and these are initially unknown
- Rewards / punishments are infrequent:
  - Often at the end of long sequences of actions
  - How do we determine what action(s) were really responsible for reward or punishment?
- World is large and complex
- Exploration vs. exploitation.

Nevertheless learner must decide what actions to take

What Methods Have We Seen So Far?

We will inspect them more closely... and also some additional approaches

Recap and what next

- Spectrum of approaches to RL
- next: deeper dive into RL methods ?