NOVA IMS Information Management School

Reinforcement Learning

Lecture 2

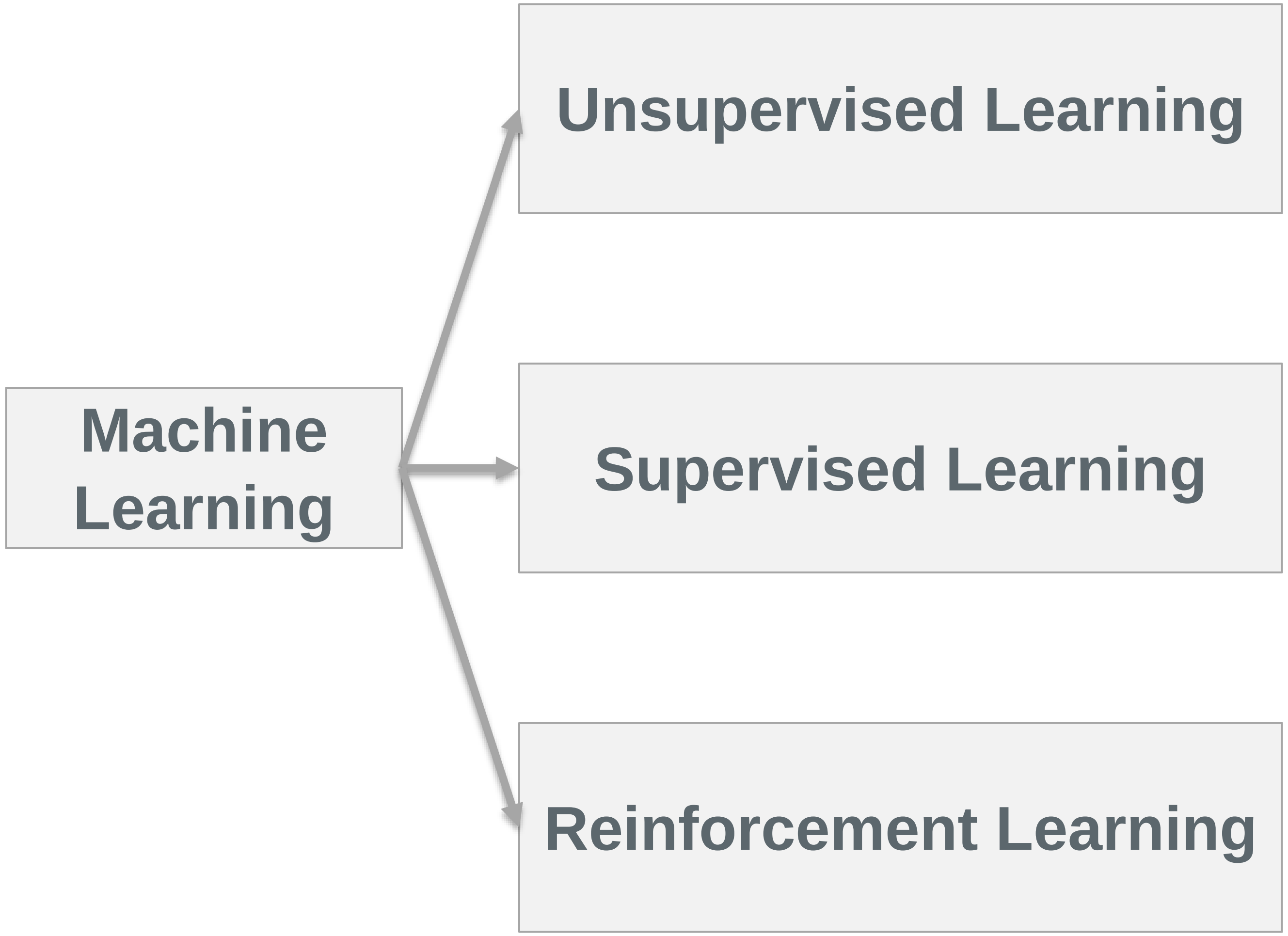
Tabular Value-Based Reinforcement Learning and Markov Decision Processes

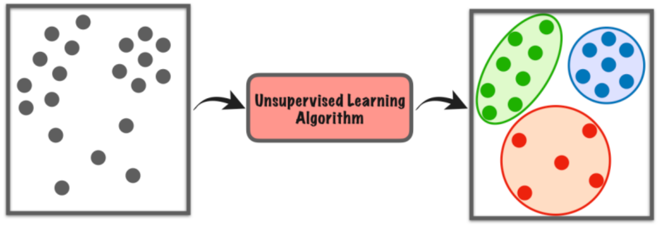
April 2025

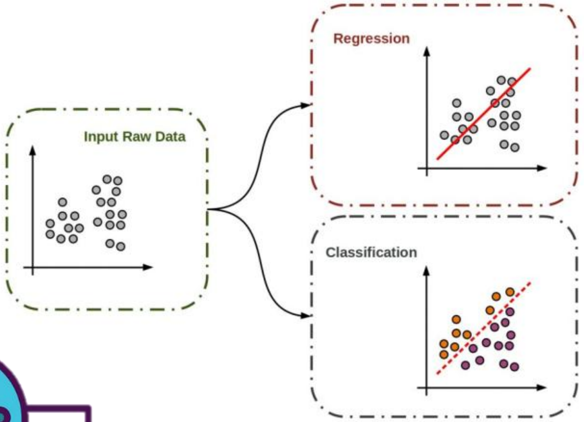
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

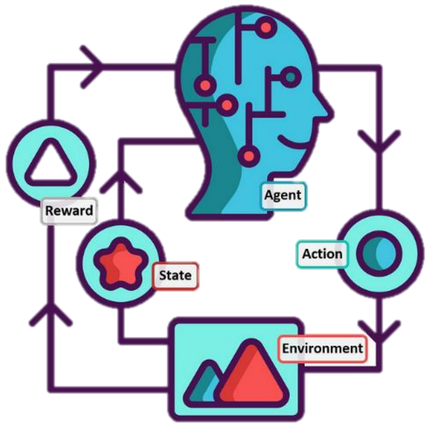
Last Class

Recap – RL in Machine Learning









Last Class

Recap – RL in Machine Learning

Reinforcement Learning Involves

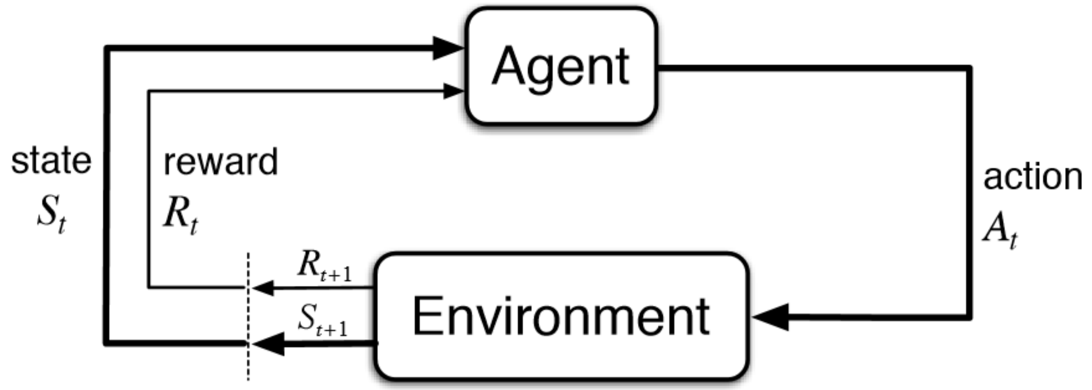
- Optimization
- Delayed consequences
- Exploration
- Generalization

Last Class Recap - Reward Hypothesis

Reinforcement learning is based on the **reward hypothesis:**

All goals can be described by the maximisation of expected cumulative reward.

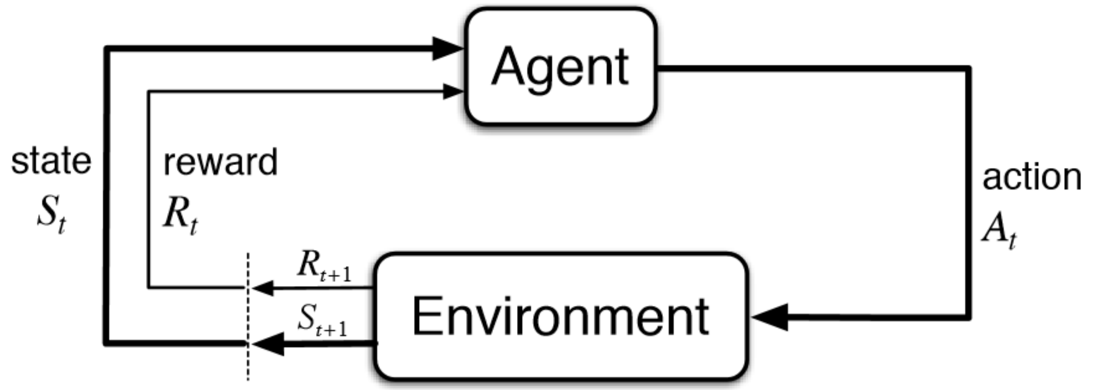
- A reward is a scalar feedback signal.
- Indicates how well agent is doing at current step.
- The agent’s job is to maximise cumulative reward.



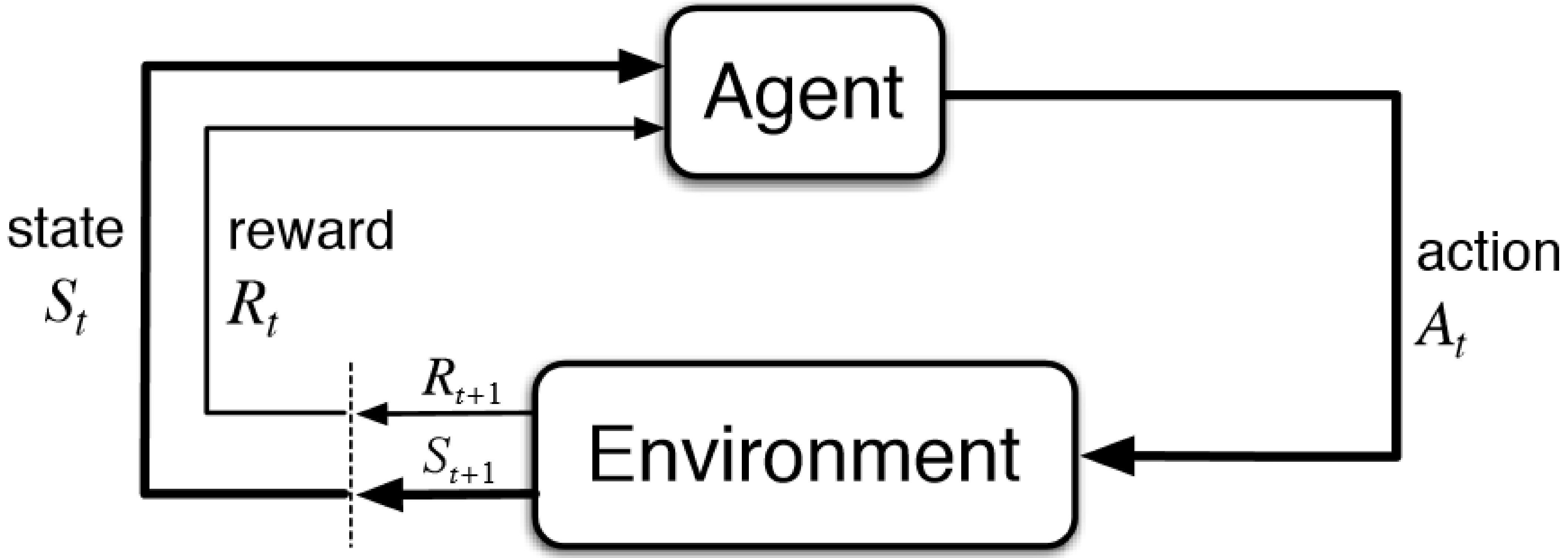
Last Class — Recap - Core Concepts

The reinforcement learning formalism includes:

- Reward signal (specifies the goal).
- Environment (dynamics of the problem).
- Agent:
  - Agent state.
  - Policy.
  - Value function.
  - Model.



RL Conceptualization

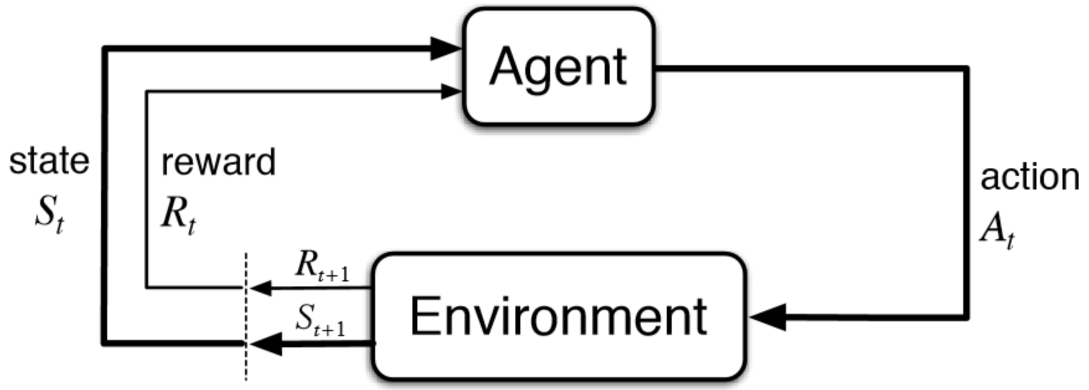


Agent and the Environment

At each step t the agent:

- Receives observation O_t (and reward R_t)
- Executes action A_t

- The environment:
  - Receives action A_t
  - Emits observation O_{t+1} (and reward R_{t+1})



Rewards

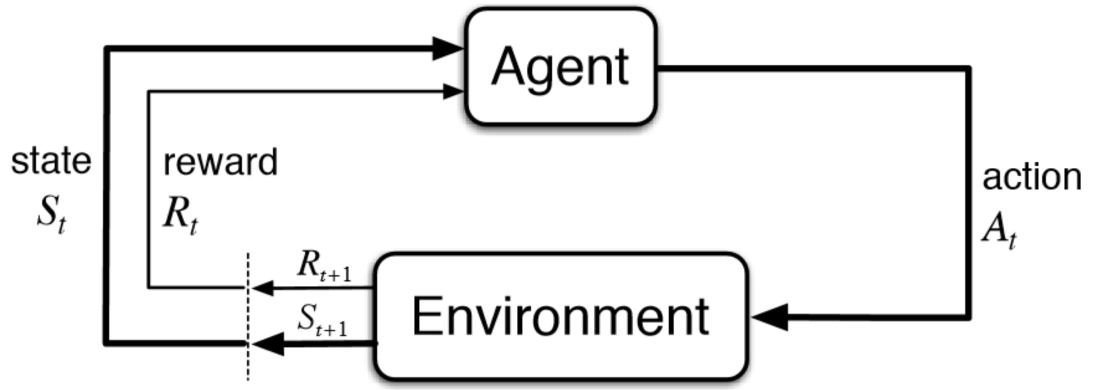
A reward R_t is a scalar feedback signal.

Indicates how well agent is doing at step *t* (defines the goal).

The agent's job is to maximize cumulative reward:

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \ \dots$$

We call this the **return**.

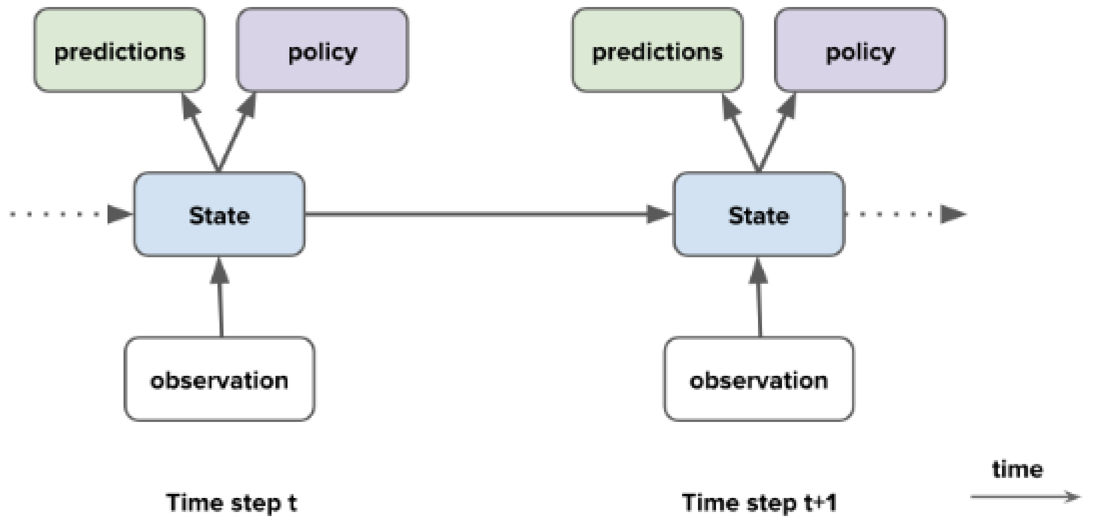


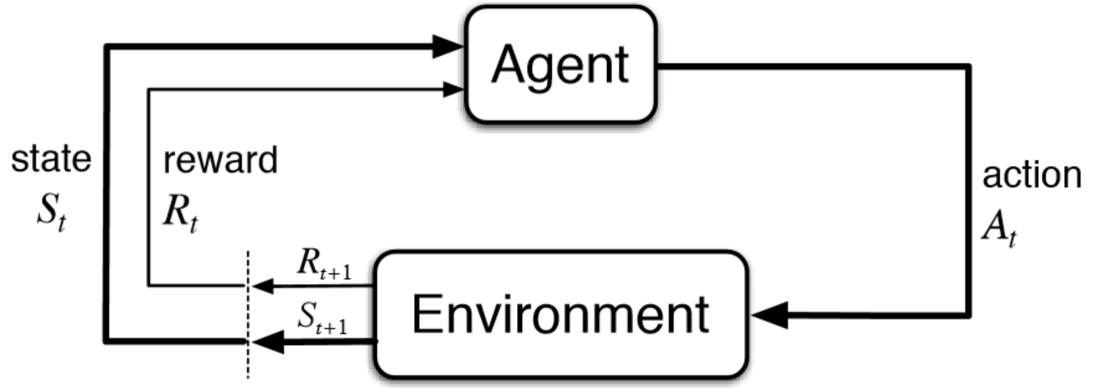
Agent State

- The history is the full sequence of observations, actions, rewards:

$$H_t = O_0, A_0, R_1, O_1, \dots, O_{t-1}, A_{t-1}, R_t, O_t$$

- This history is used to construct the **agent state** S_t.





Agent State

The agent's actions depend on its state.

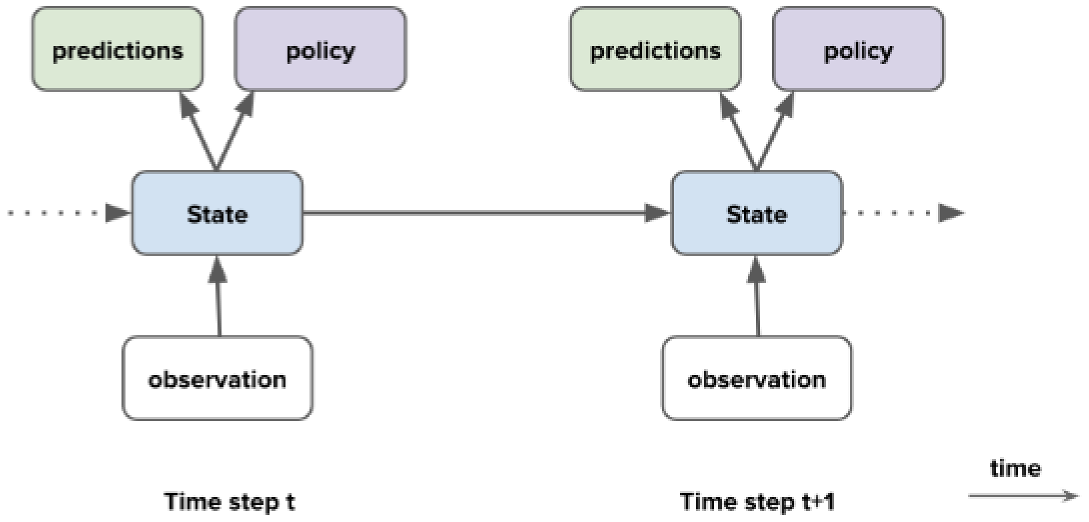
The agent state is a function of the history.

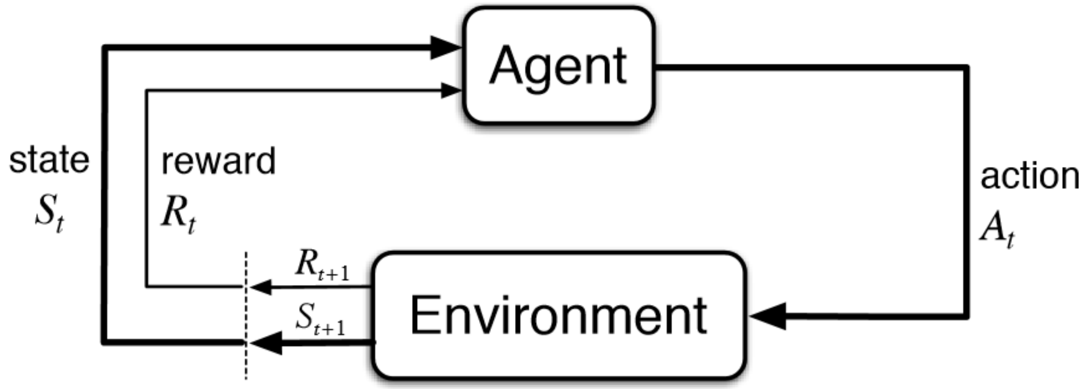
More generally:

$$S_{t+1} = u(S_t, A_t, R_{t+1}, O_{t+1})$$

Where u is a '**state update function**'

The agent state is often much smaller than the environment state.



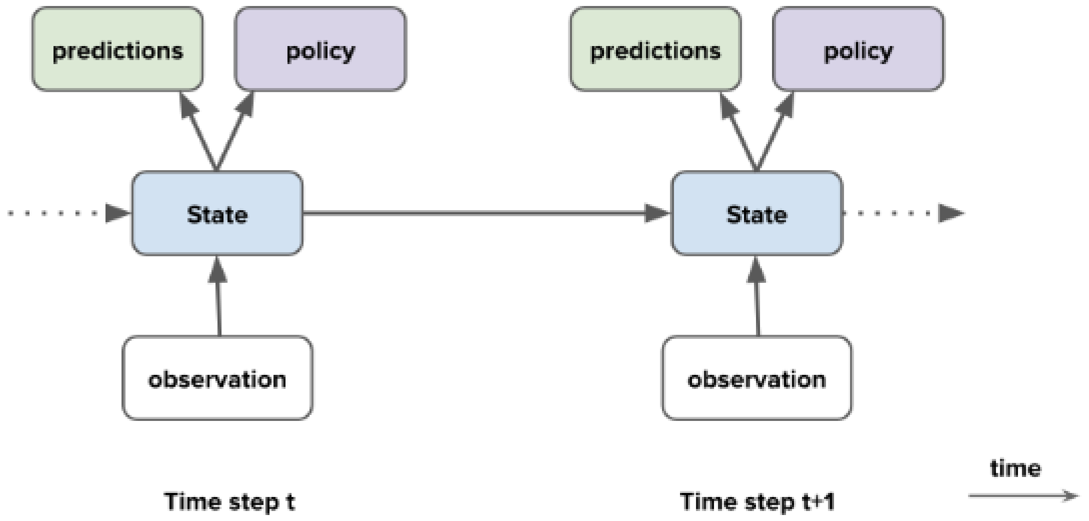


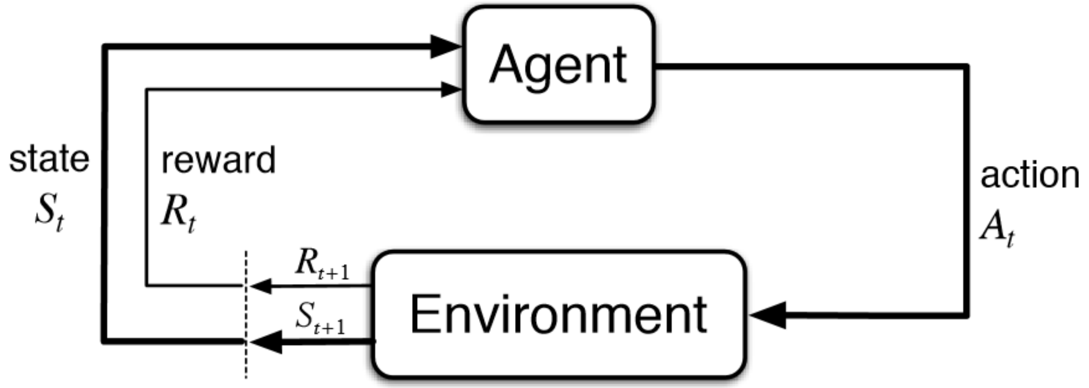
Agent Policy

A policy defines the agent's behaviour

It is a **map from agent state to action**:

- **Deterministic policy**: $A = \pi(S)$
- **Stochastic policy**: $\pi(A \mid S) = p(A \mid S)$





Model

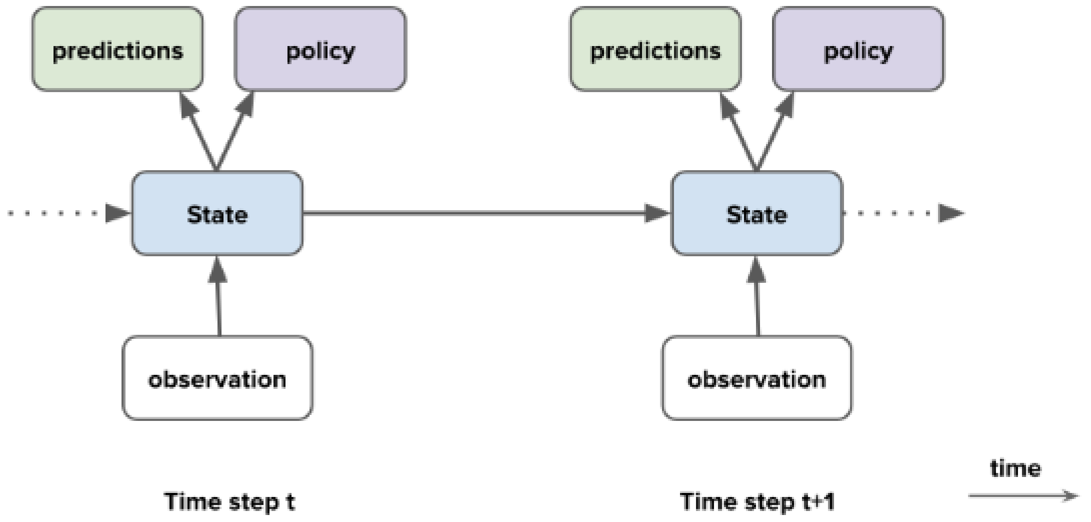
Agent's representation of how world changes given agent's action.

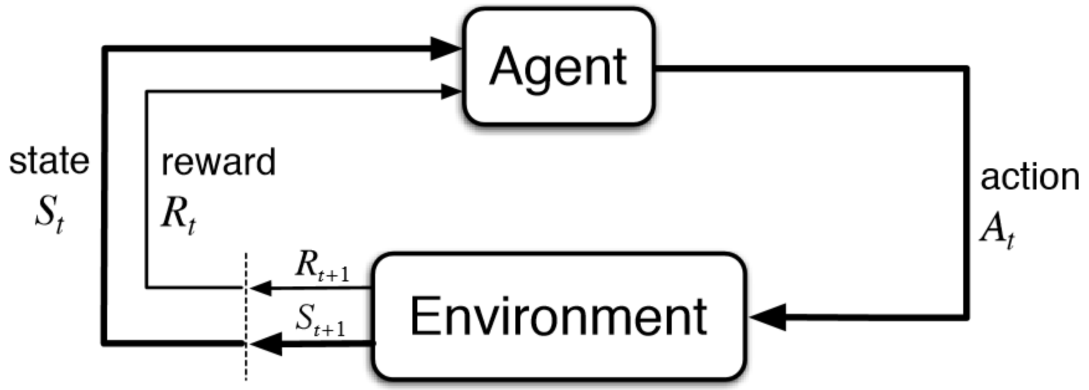
Transition / dynamics model P predicts the next state:

$$P(s,a,s') \approx p\,(S_{t+1} = s' \mid S_t = s, A_t = a)$$

Reward model R predicts the next (immediate) reward:

$$R(s,a) \approx E\,[R_{t+1} \mid S_t = s, A_t = a]$$

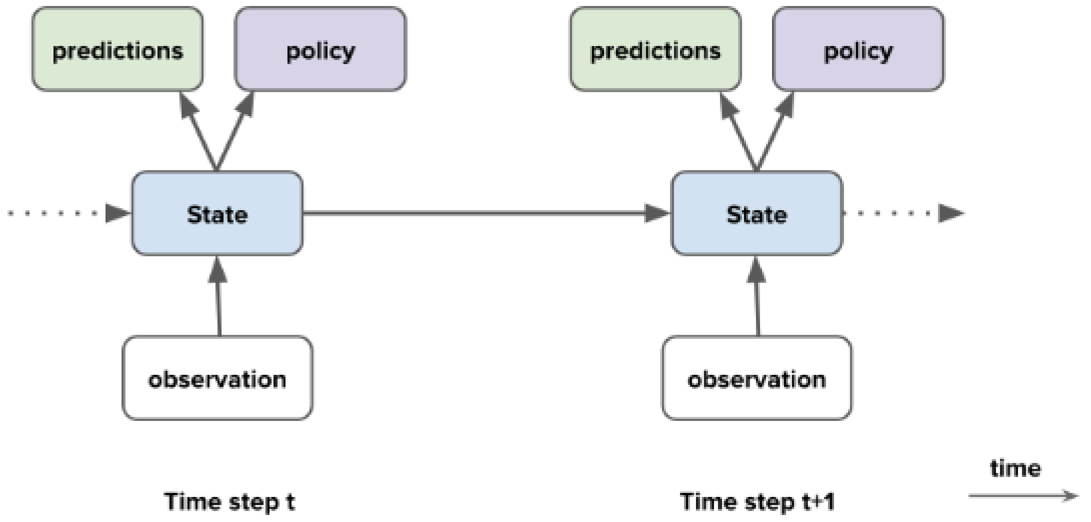


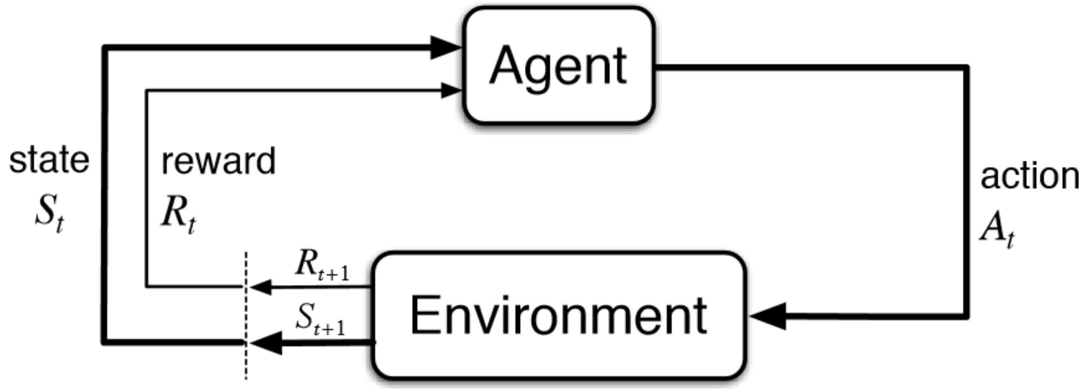


Agent Model

A model does not immediately give us a good policy - we would still need to plan.

We could also consider stochastic (generative) models.



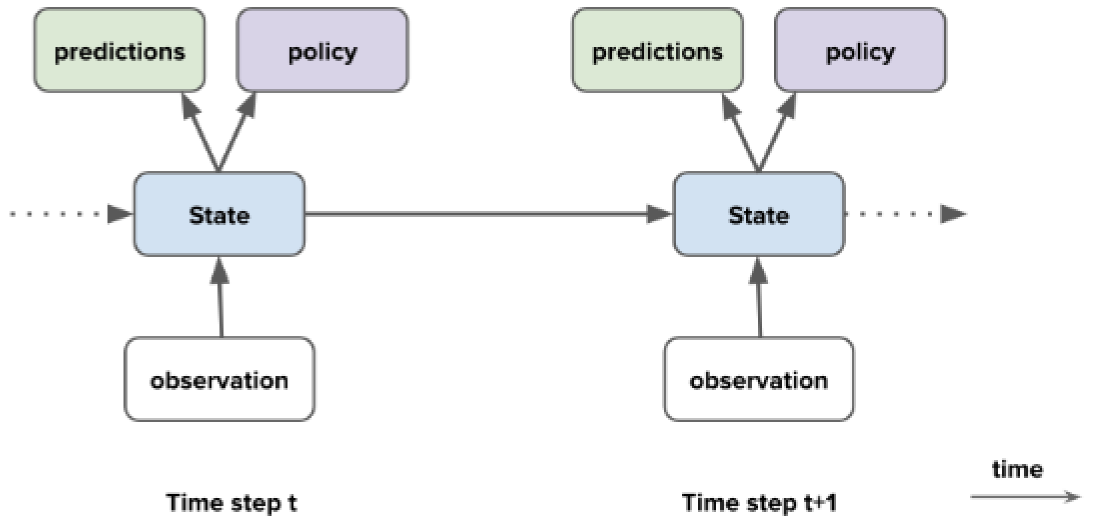


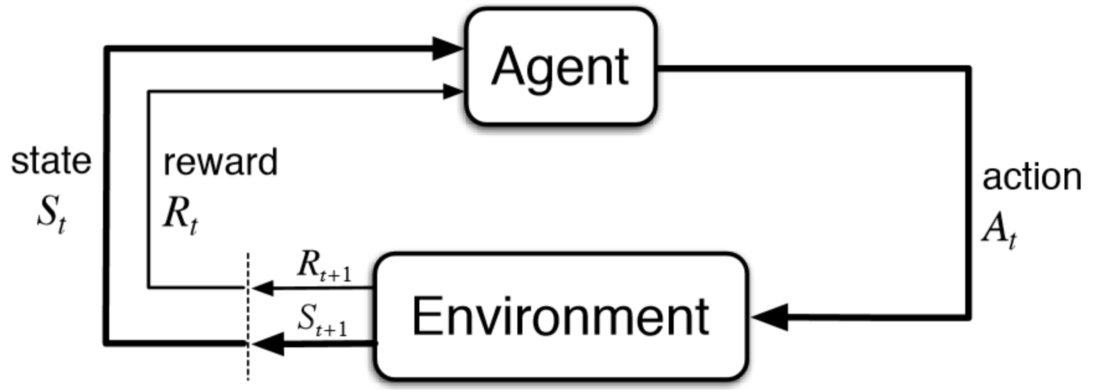
Agent Value Function

The actual value function is the expected cumulative rewards or **expected return**:

$$v_\pi(s) = E\,[G_t \mid S_t = s, \pi\,]$$
$$= E\,[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ \dots \mid S_t = s, \pi\,]$$

We introduced a discount factor $\gamma \in [0,1]$.





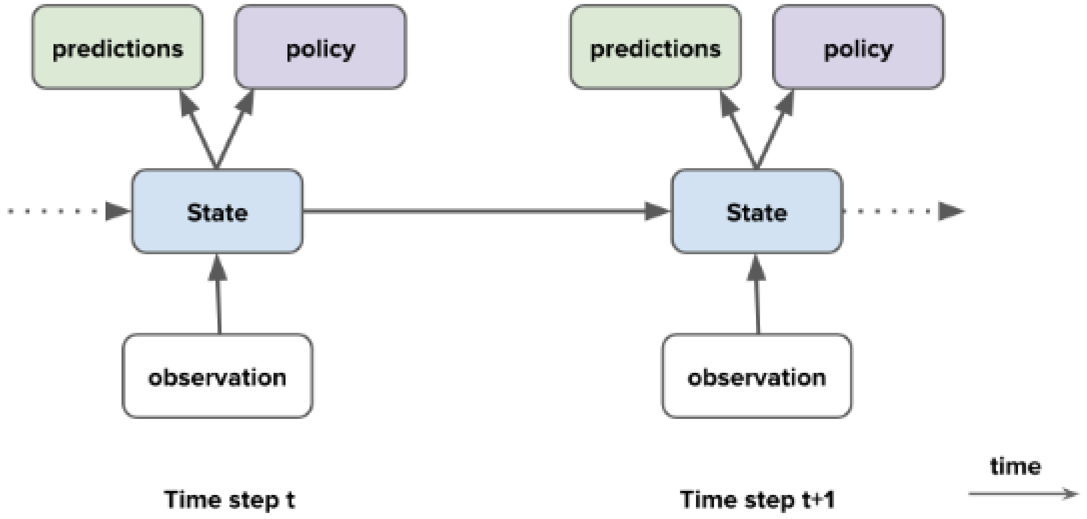
Agent Value Function

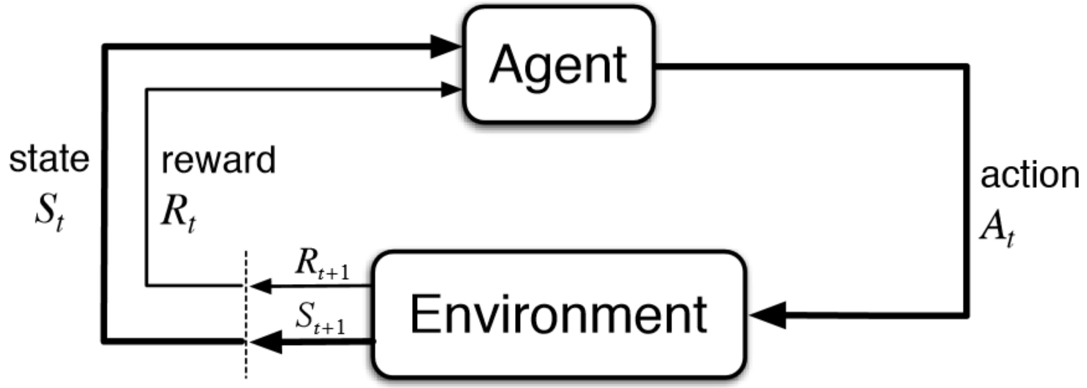
Discount factor $\gamma \in [0,1]$.

Trades off importance of immediate vs long-term rewards.

The value depends on a policy:

- Can be used to evaluate the desirability of states.
- Can be used to select between actions.





Agent Value Function

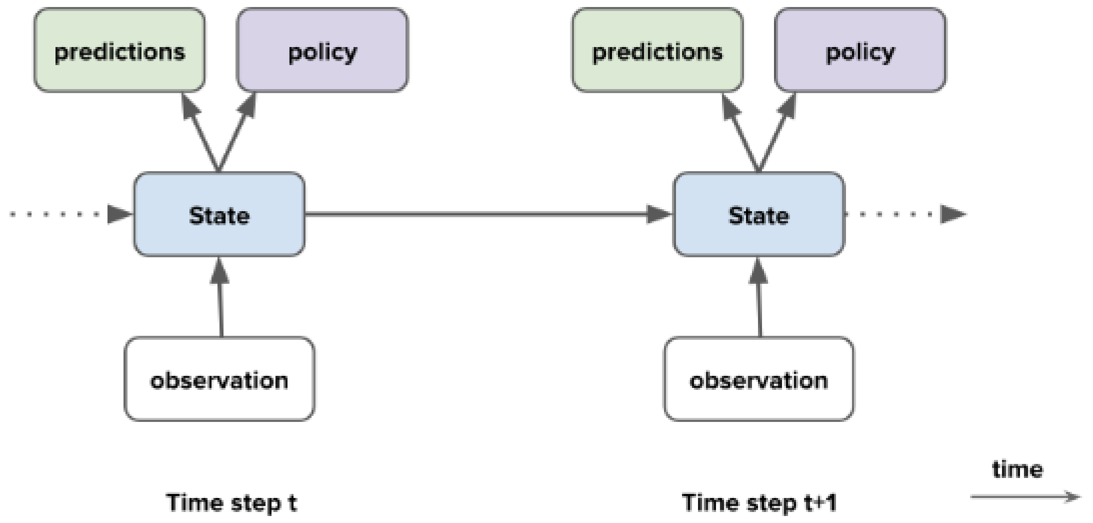
The return has a recursive form:

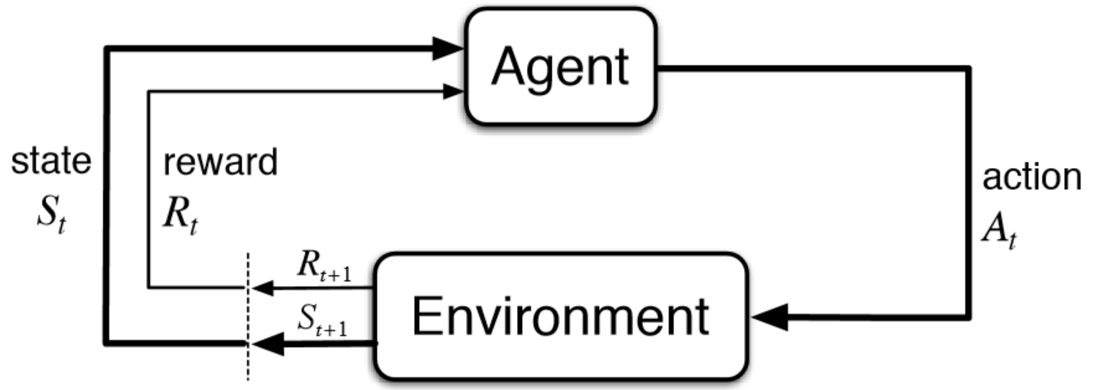
$$G_t = R_{t+1} + \gamma G_{t+1}.$$

Therefore, the value has as well:

$$\begin{aligned} v_\pi(s) &= E\left[G_t \mid S_t = s, \pi\right] \\ &= E\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t \sim \pi\right] \\ &= E\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t \sim \pi\right] \end{aligned}$$

Here $a \sim \pi(s)$ means action a is chosen by policy π in state s (even if π is deterministic).





Agent Value Function

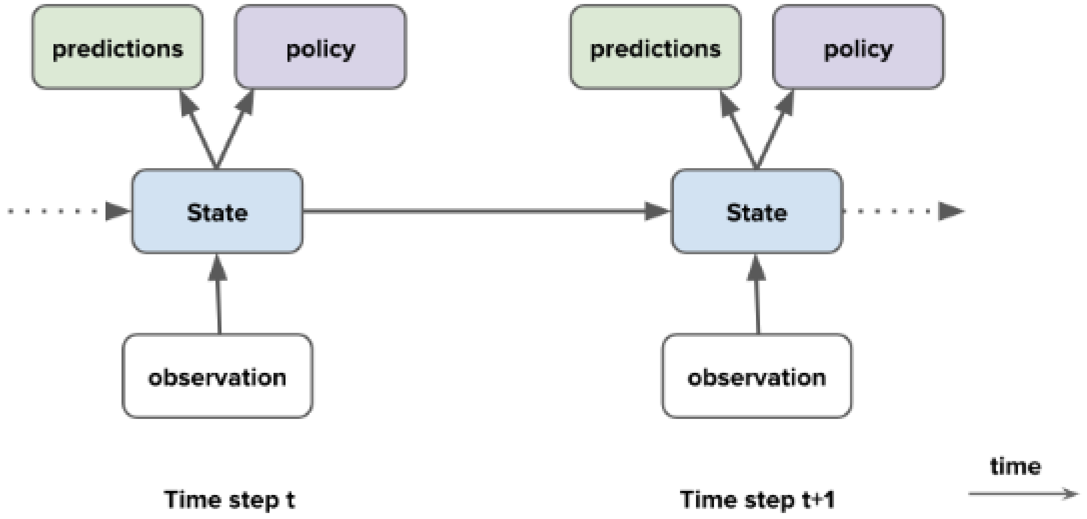
This is known as a **Bellman equation** (Bellman 1957).

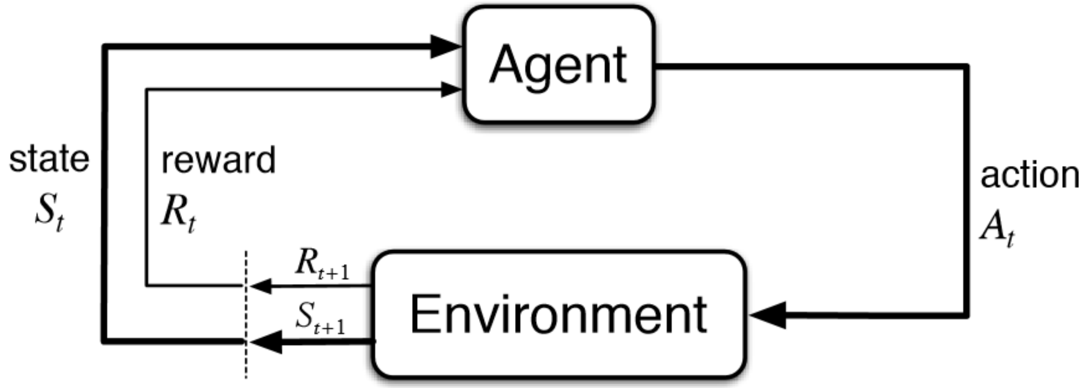
A similar equation holds for the optimal (highest possible) value:

$$v_* (s) = \max_a E \left[R_{t+1} + \gamma v_\pi (S_{t+1}) \mid S_t = s, A_t = a\right]$$

This does not depend on a policy.

We heavily exploit such equalities and use them to create algorithms.





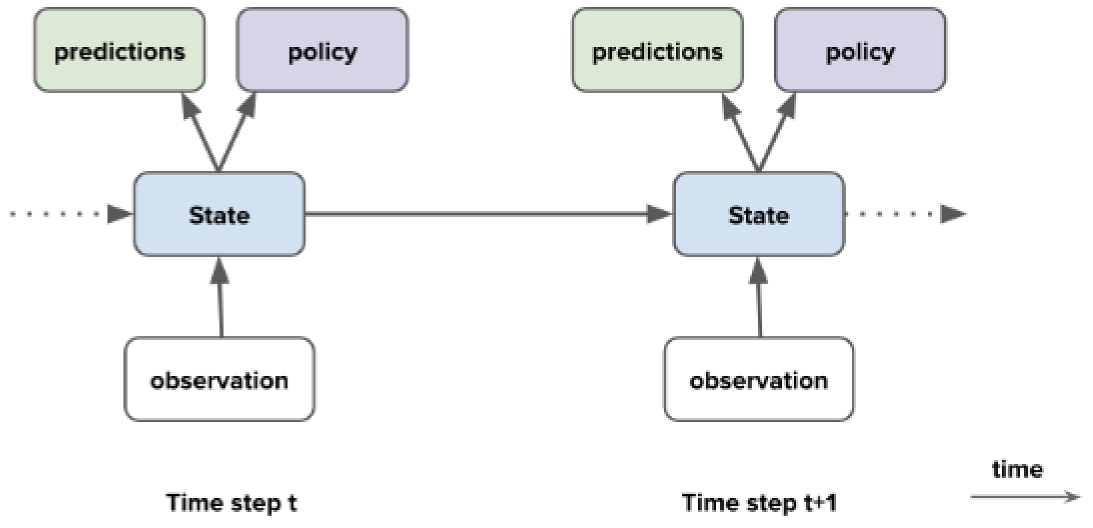
Agent Value Function

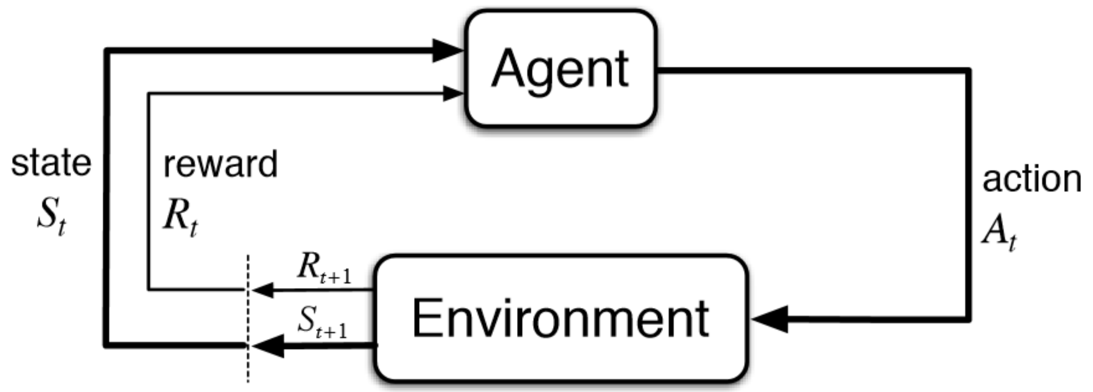
Agents often approximate value functions.

We will discuss algorithms to learn these efficiently.

With an accurate value function, we can behave optimally.

With suitable approximations, we can behave well, even in intractably big domains.





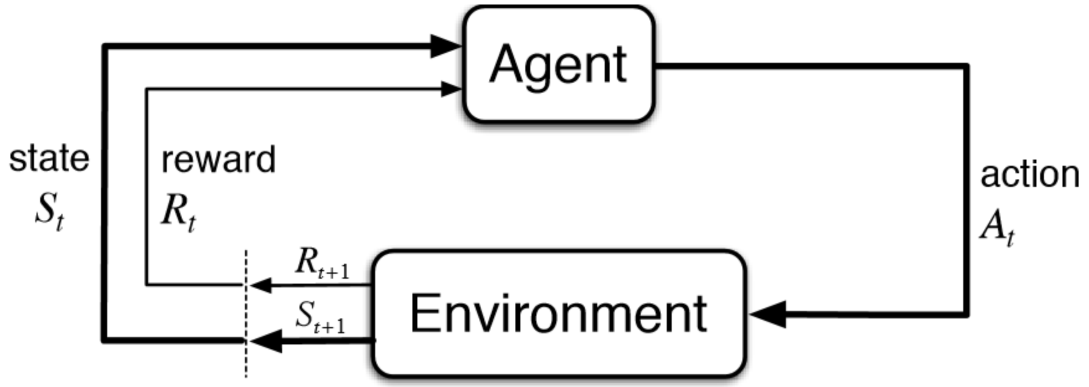
Action Values

It is also possible to condition the value on actions:

$$Q(s,a) = E\left[G_t | S_t = s, A_t = a\right]$$

$$= E\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots | S_t = s, A_t = a\right]$$

This type of value function is called a ***Q*** **function.**



Environment State

Markov decision processes (MDPs) are a useful mathematical framework:

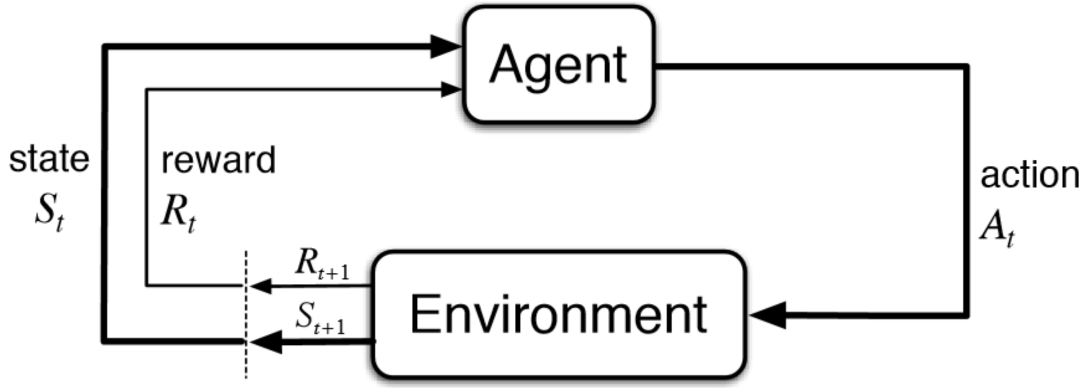
Definition

A decision process is Markov if:

$$p\,(r, s\,|S_t\,,\,A_t\,)\;=\;p\,(r, s\,|\,H_t, A_t\,)$$

This means that the state contains all we need to know from the history.

Doesn't mean it contains everything, just that adding more history doesn't help.

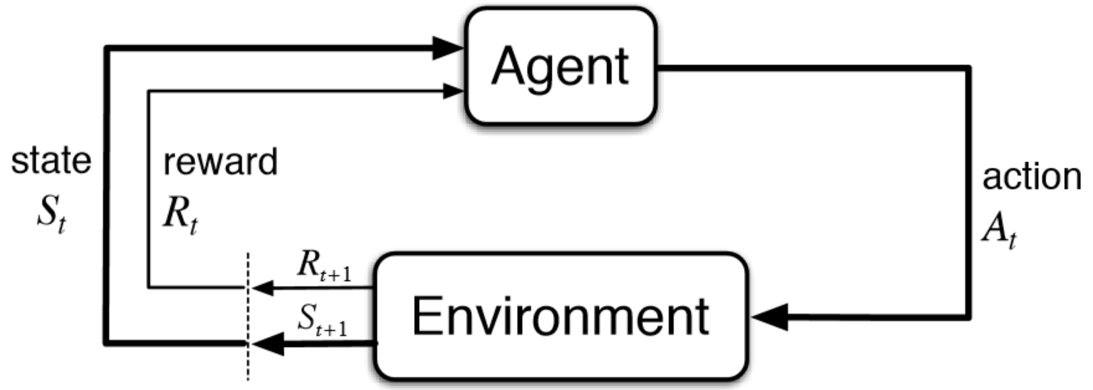


Full observability

Suppose the agent sees the full environment state:

- observation = environment state
- The agent state could just be this observation:

$$S_t = O_t = \text{environment state}$$



Environment State

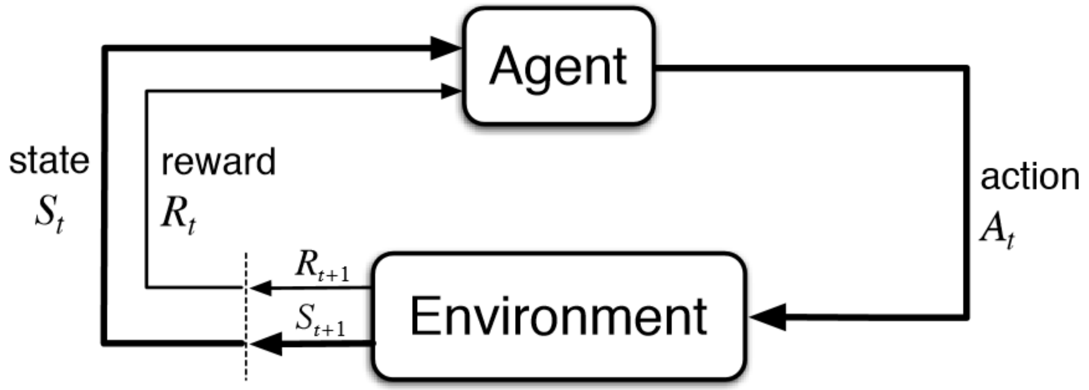
Partial observability: The observations are not Markovian

Now using the observation as state would not be Markovian.

This is called a **partially observable Markov decision process** (POMDP).

The environment state can still be Markov, but the agent does not know it.

We might still be able to construct a Markov agent state.

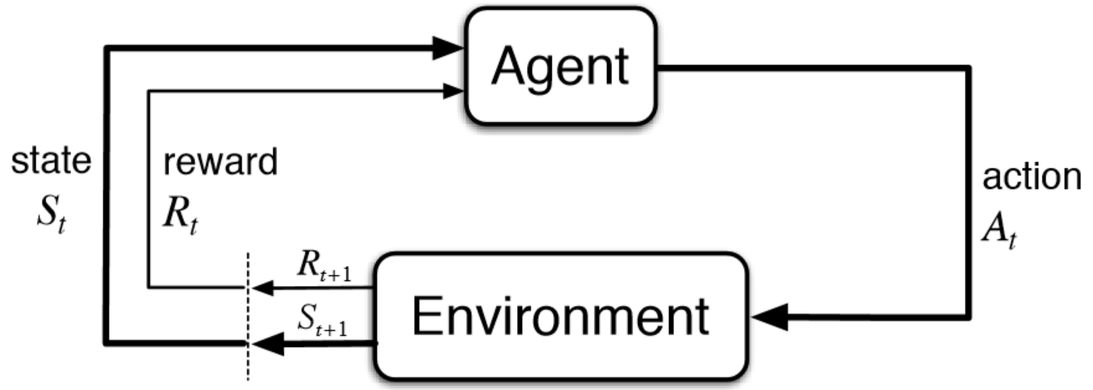


Environment State

Once the state is known, the history may be thrown away.

Typically, the agent state S_t is some compression of H_t.

Note: we use S_t to denote the agent state, not the environment state.



Exploring MDP's

"The future is independent of the past given the present"

Definition

A state S_t is Markov if and only if:

$$P\,[\,S_{t+1}|\,S_t] \;=\; P\,[\,S_{t+1}|\,S_1,\dots,S_t\,]$$

The state captures all relevant information from the history.

Once the state is known, the history may be thrown away.

The current state is a sufficient statistic of the future.

MDP’s - State Transition Matrix

For a Markov state s and successor state s′, the state transition probability is defined by:

$$P_{ss'} = P\,[\,S_{t+1} = s' \mid S_t = s]$$

State transition matrix P defines transition probabilities from all states s to all successor states s′:

$$P = \begin{bmatrix} P_{11} & \cdots & P_{1n} \\ \vdots & \ddots & \vdots \\ P_{n1} & \cdots & P_{nn} \end{bmatrix}$$

Where each row of the matrix sums to 1.

MDP's - State Transition Matrix

A Markov process is a memoryless random process, i.e. a sequence of random states S_1, S_2, ... with the Markov property.

Definition

A Markov Process (or Markov Chain) is a tuple $\langle S, P \rangle$

- S is a (finite) set of states.
- P is a state transition probability matrix:

$$P_{ss'} = P\,[\,S_{t+1} = s' \mid S_t = s]$$

Markov Reward Process

A Markov reward process is a Markov chain with values.

Definition

A Markov Reward Process is a tuple $\langle S, P, R, \gamma \rangle$

- S is a finite set of states
- P is a state transition probability matrix:

$$P_{ss'} = P\left[S_{t+1} = s' \mid S_t = s \right]$$

Markov Reward Process

A Markov reward process is a Markov chain with values.

Definition

A Markov Reward Process is a tuple $\langle S, P, R, \gamma \rangle$

- R is a reward function:

$$R(s,a) \approx E\,[\,R_{t+1} \mid S_t = s, A_t = a]\;(Part\ of\ the\ agent's\ model)$$

$$R(s,a,s') \approx E\,[\,R_{t+1} \mid S_t = s, A_t = a, S_{t+1} = s']\;(**\ reward\ probability)$$

- γ is a discount factor, $\gamma \in [0,1]$

MRP's Horizon

Definition of Horizon (H)

Definition of **Return**, G_t (for a MRP)

- Discounted sum of rewards from time step *t* to horizon H

Definition of **State Value Function, $V(s)$** (for a MRP)

- Expected return from starting in state s:

$$V(s) = E[\,G_t \mid S_t = s]$$

MRP's Horizon

Definition of Horizon (H)

- Number of time steps in each episode.
- Can be infinite.
- Otherwise called finite Markov reward process.
- For finite state MRP, we can express $V(s)$ using a matrix equation

State-action value function (Q function)

A state-action value function can also be called the Q function. It specifies how good it is for an agent to perform a particular action in a state with a policy π.

The Q function is denoted by $Q(s, a)$.

It denotes the expected value of taking an action in a state following a policy π.

We can define Q function as follows:

$$Q^{\pi}(s, a) = E_{\pi}\left[G_t \mid S_t = s, A_t = a\right]$$

State-action value function (Q function)

This specifies the expected return in state s with the action a according to policy π.

The difference between the value function and the Q function is that the value function specifies the goodness of a state, while a Q function specifies the goodness of an action in a state.

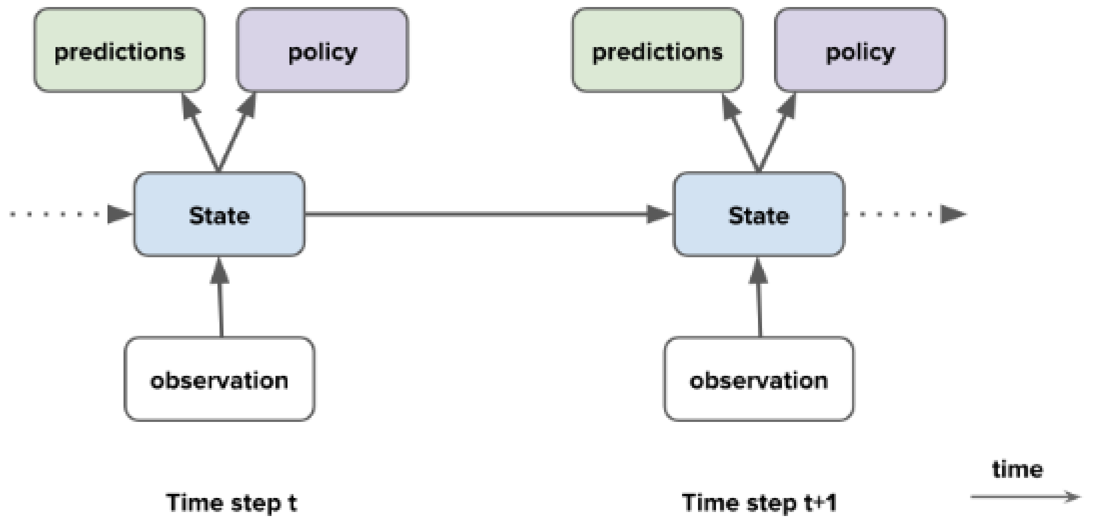
Agent Value Function

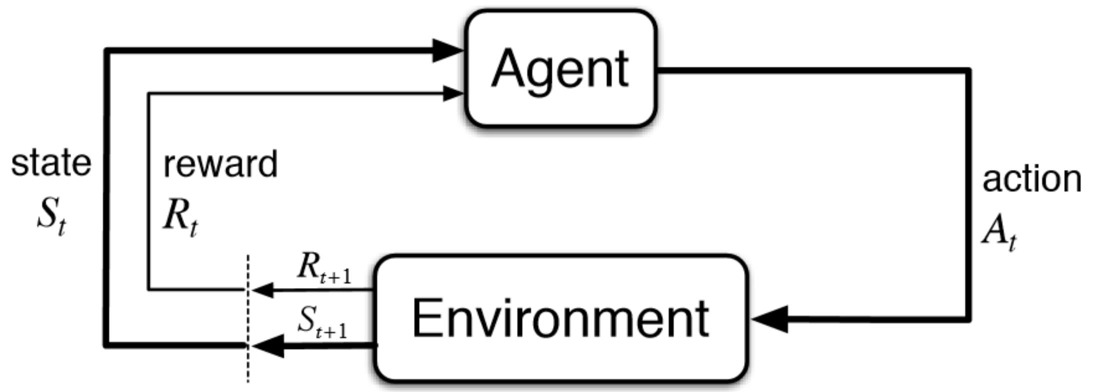
The return has a recursive form $G_t = R_{t+1} + \gamma G_{t+1}$.

Therefore, the value has as well:

$$\begin{aligned} v_\pi(s) &= E\,[G_t \mid S_t = s, \pi\,] \\ &= E\,[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t \sim \pi\,] \\ &= E\,[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t \sim \pi\,] \end{aligned}$$

Here $a \sim \pi(s)$ means action a is chosen by policy π in state s (even if π is deterministic).





Agent Value Function

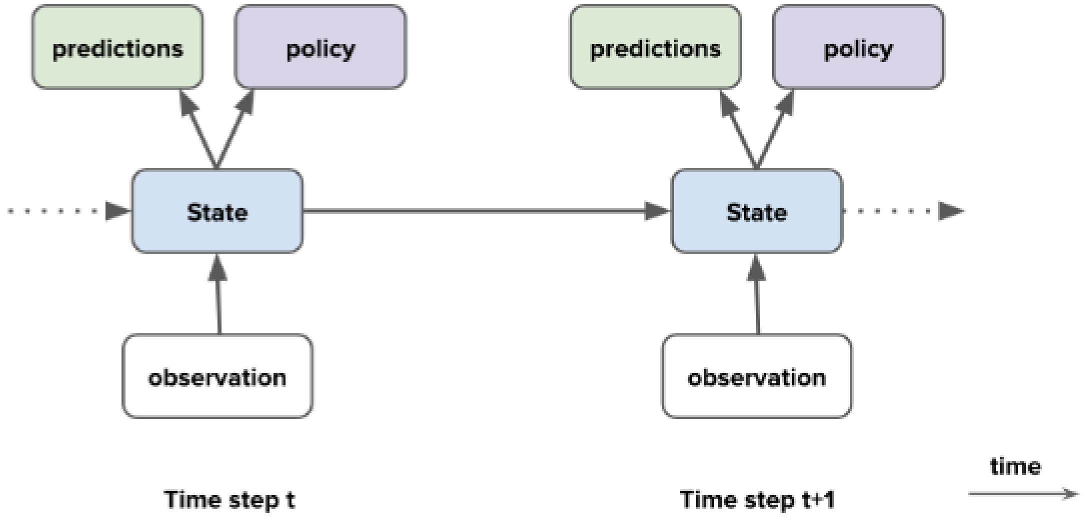
This is known as a **Bellman equation** (Bellman 1957).

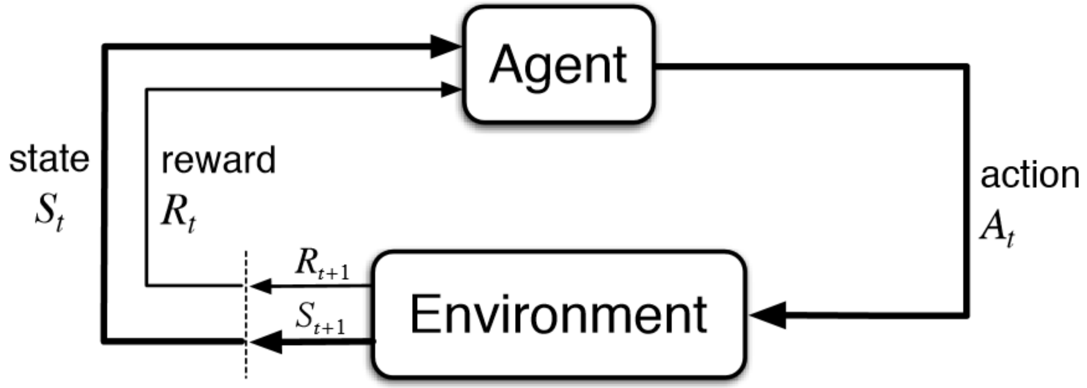
A similar equation holds for the optimal (highest possible) value:

$$v_*(s) = max_a E[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a]$$

This does not depend on a policy.

We heavily exploit such equalities and use them to create algorithms.





Agent Value Function

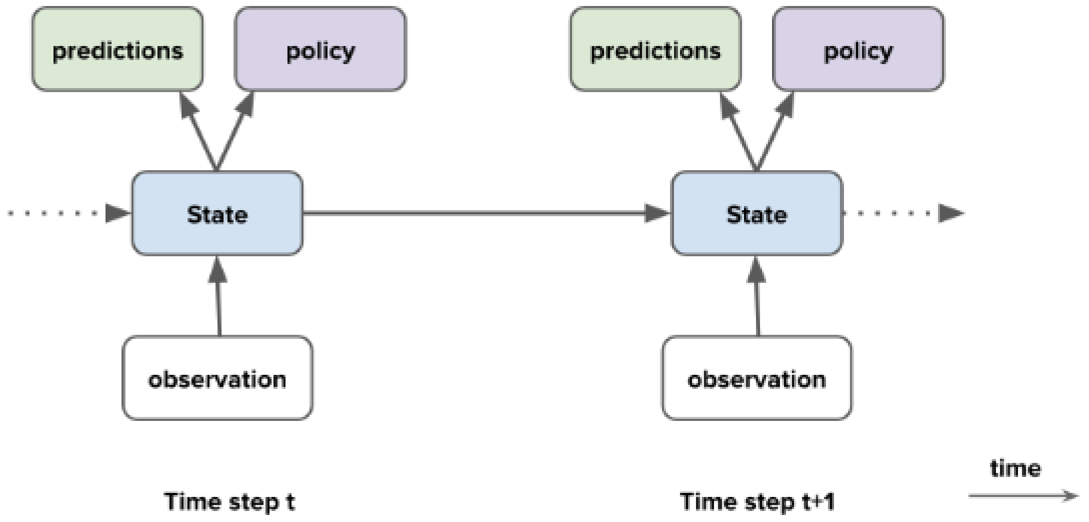
In another formulation, the **Bellman** equation allows the solution of the MDP:

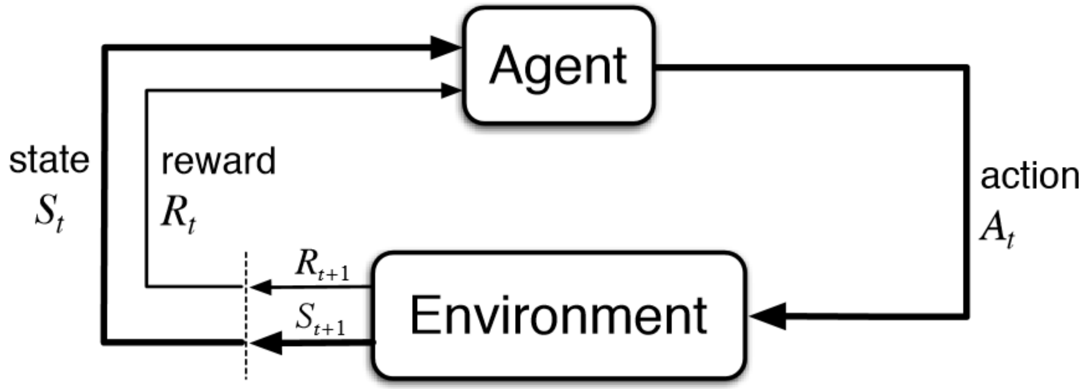
$$v_*(s) = max_\pi v_\pi(s)$$

It means finding the optimal policies and value functions.

There can be many different value functions according to different policies.

The optimal value function is the one which yields maximum value compared to all the other value functions.





Bellman Expectation Equations

Given an MDP, $M = \langle S, A, p, r, \gamma \rangle$, the optimal value functions obey the following expectation equations:

$$v_\pi (s) = \sum_a \pi(a,s) \sum_{s'} p(s'|a,s)\, [R(s,a,s') + \gamma v_\pi (s')]$$

$$Q^\pi(s,a) = \sum_{s'} p(s'|a,s)[R(s,a,s') + \gamma \sum_{s'} Q^\pi(s',a')]$$

There can be no policy with a higher value than:

$$\forall s : \; v_* (s) = \; max_\pi v_\pi (s)$$

Bellman Optimality Equations

Given an MDP, $M = \langle S, A, p, r, \gamma \rangle$, the optimal value functions obey the following expectation equations:

$$v_* (s) = max_a [R(s,a) + \gamma \sum_{s'} p(s'|a,s) \, v_* (s')]$$

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

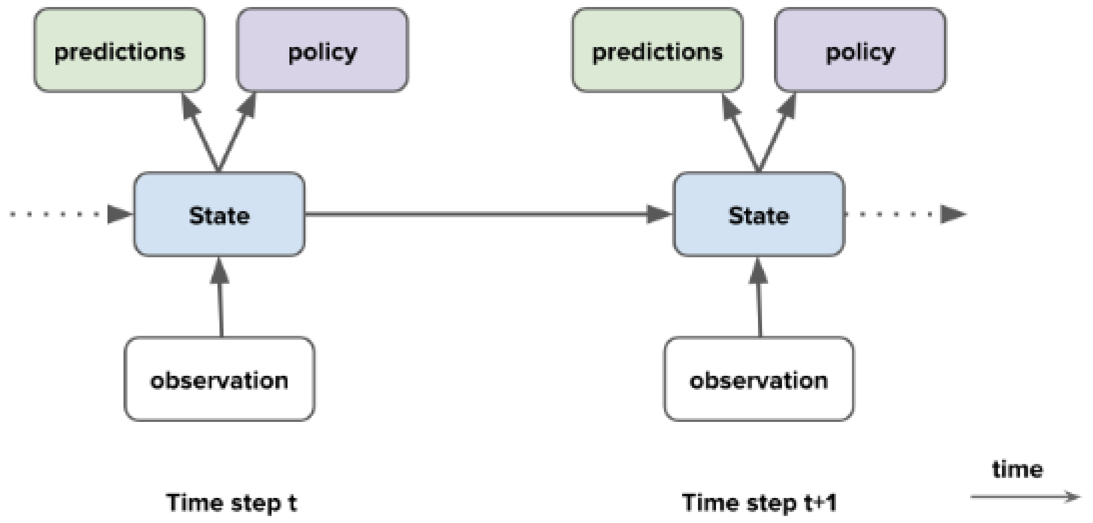
There can be no policy with a higher value than:

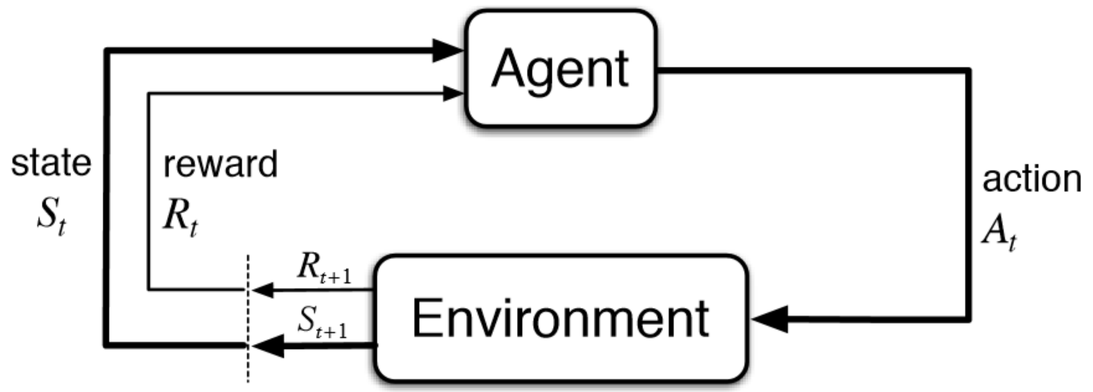
$$\forall s : \; v_* (s) = \; max_\pi v_\pi (s)$$

Agent Categories I

Agent Categories

- Value Based
  - No Policy (Implicit)
  - **Value Function**
- Policy Based
  - **Policy**
  - No Value Function
- Actor Critic
  - **Policy**
  - **Value Function**

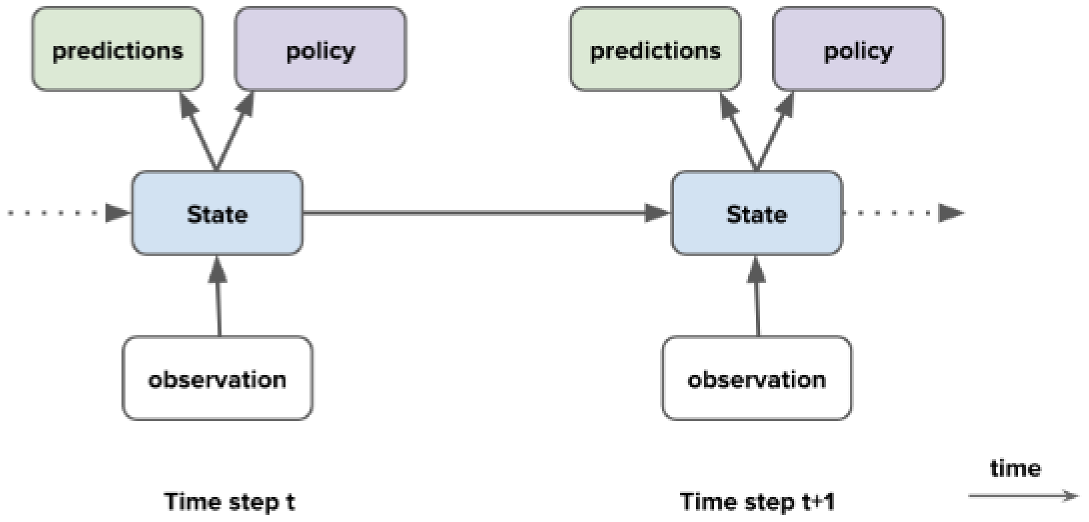


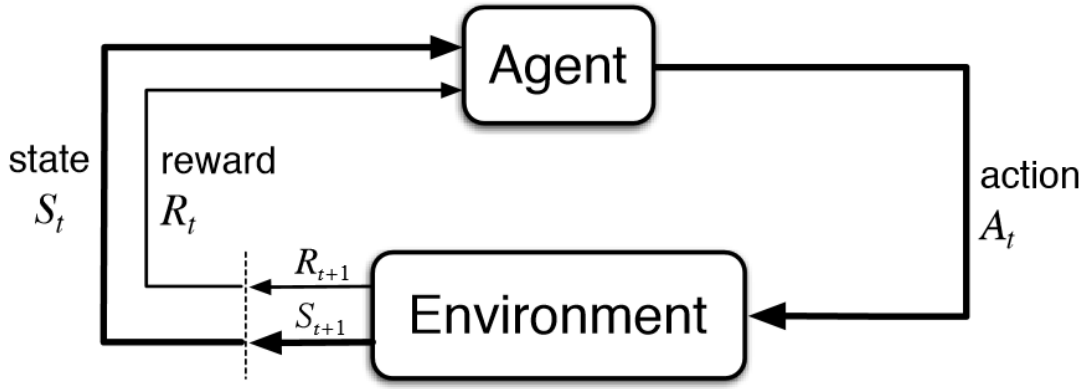


Agent Categories II

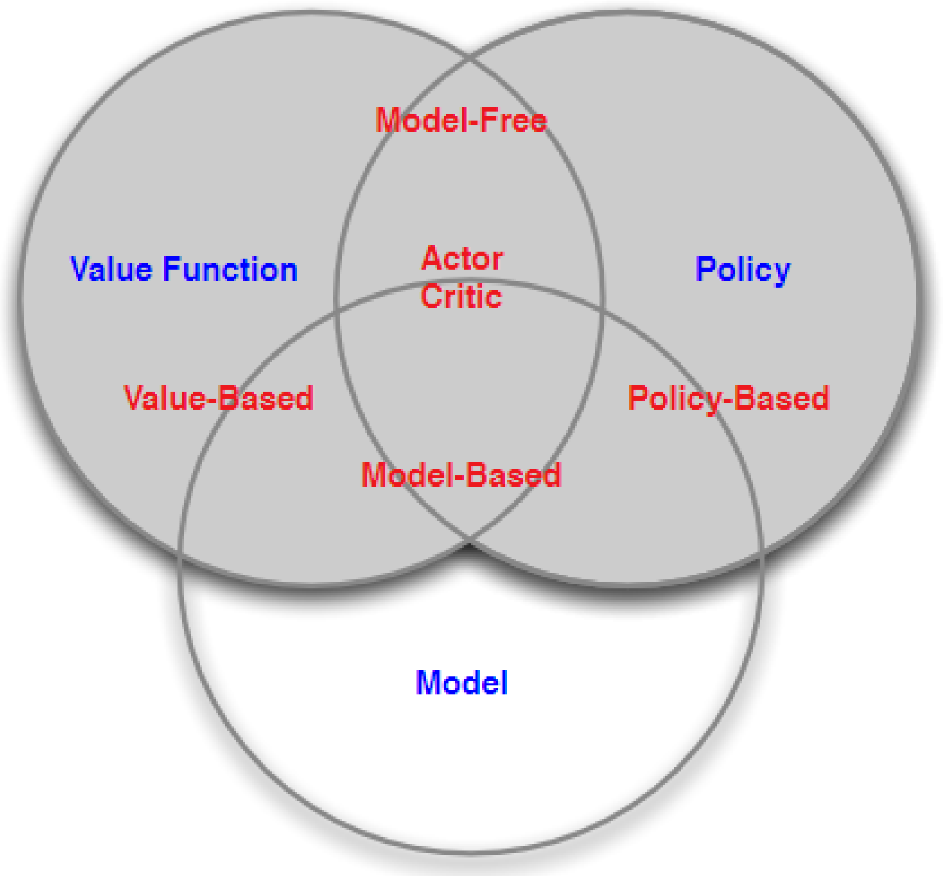
Agent Categories

- Model Free
  - **Policy and/or Value Function**
  - No Model
- Model Based
  - Optionally Policy and/or Value Function
  - **Model**





Many Concepts of RL



From DeepMind’s and UCL with David Silver Course

Prediction/Evaluation vs Control

Prediction: evaluate the future (for a given policy)

Control: optimise the future (find the best policy)

- These can be strongly related:

$$\pi_*(s) = argmax_\pi \, v_\pi(s)$$

- If we could predict everything, do we need anything else?

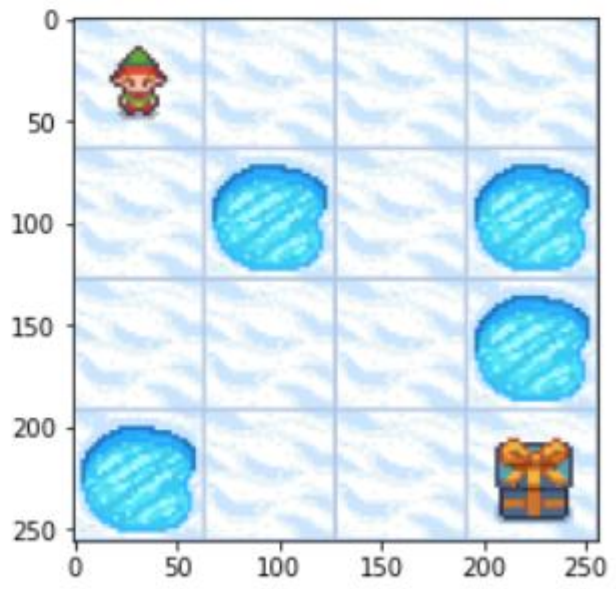
Agent's Functions Overview

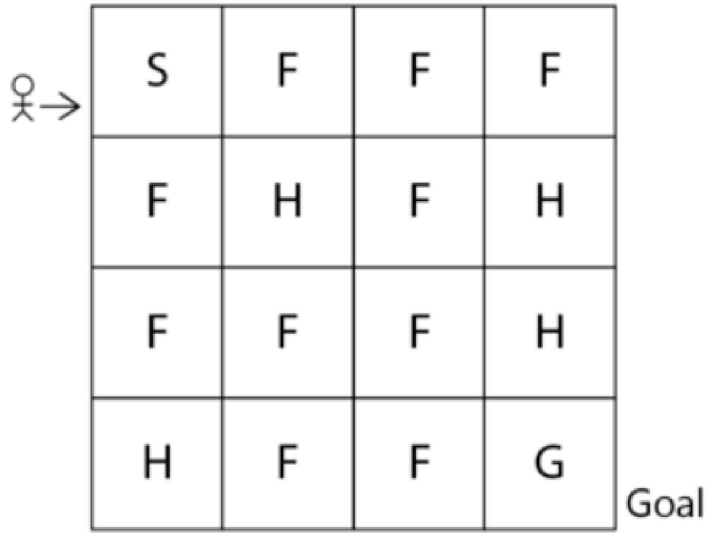
- All components are functions
  - **Policies** $\pi : S \rightarrow A$ (or to probabilities over A)
  - **Value functions** $v : S \rightarrow R$
  - **Models** $m : S \rightarrow S$ and/or $r : S \rightarrow R$
  - **State update** $u : S \times O \rightarrow S$

- Often the **assumptions from supervised learning** (i.i.d., stationarity) are disrupted.

Frozen Lake example

- Imagine there is a frozen lake stretching from your home to your office; you have to walk on the frozen lake to reach your office. But oops! There are holes in the frozen lake so you have to be careful while walking on the frozen lake to avoid getting trapped in the holes:





Frozen Lake example

In the above environment, the following applies:

- **S** denotes the starting state
- **F** denotes the frozen state
- **H** denotes the hole state
- **G** denotes the goal state

So, the agent has to start from the state **S** and reach the goal state **G**. But one issue is that if the agent visits the state **H**, which is just the hole state, then the agent will fall into the hole and die.

Frozen Lake example

We have:

- **States** - A set of states present in the environment.
- **Actions** - A set of actions that the agent can perform in each state.
- **Reward function** - Reward function is denoted by $R(s, a, s')$. It implies the reward the agent obtains moving from a state s to the state s' while performing an action a.
- **Transition probability** - The transition probability is denoted by $p(s'|a, s)$. It implies the probability of moving from a state s to the state s' while performing an action a.

Example - Agent Model

Agent Model

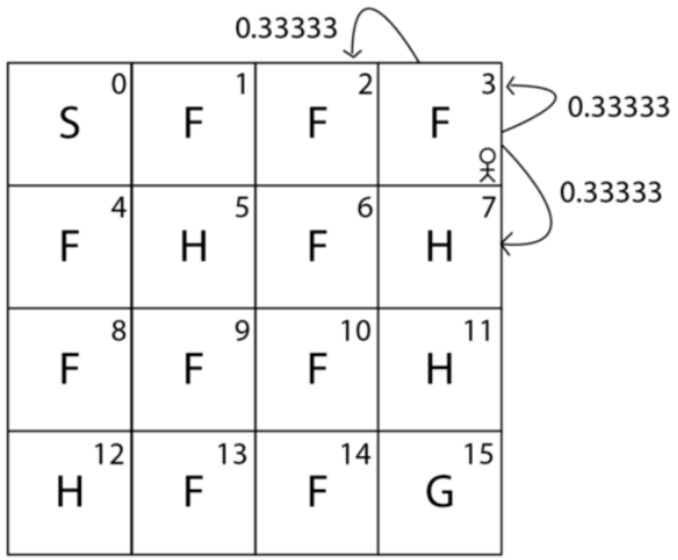
A model predicts what the environment will do next.

P predicts the next state:

$$P(s,a,s') \approx p\,(S_{t+1} = s' \mid S_t = s, A_t = a)$$

R predicts the next (immediate) reward:

$$R(s,a) \approx E\,[R_{t+1} \mid S_t = s, A_t = a]$$



Transition Probability	Next State	Reward	Is terminal State
0.33333	2(F)	0.0	False
0.33333	7(H)	0.0	True
0.33333	3(F)	0.0	False

Frozen Lake example

Now our objective is to solve MDP. Solving the MDP implies finding the optimal policies.

We still need to optimize three functions:

- **Policy function**: Specifies what action to perform in each state.
- **Value function**: Specifies how good a state is.
- **Q function**: Specifies how good an action is in a particular state.

Frozen Lake example

When we say how good, what does that really mean? It implies how good it is to maximize the rewards.

We’ll represent the **value function** and **Q function** using the Bellman Optimality equation. If we solve this equation, we can find the optimal policy.

Here, solving the equation means finding the right value function and policy. If we find the right value function and policy, that will be our optimal path which yields maximum rewards.

To solve the **Bellman optimality equation,** we will use a technique called **dynamic programming**.

Next Class

Dynamic programming

Dynamic programming (DP) is a technique for solving complex problems. In DP, instead of solving complex problems one at a time, we break the problem into simple sub-problems, then for each sub-problem, we compute and store the solution. If the same sub-problem occurs, we will not recompute, instead, we use the already computed solution.

Thus, DP helps in drastically minimizing the computation time. It has its applications in a wide variety of fields including computer science, mathematics, bioinformatics, and so on.

We solve a **Bellman equation** using two algorithms from DP:

- **Value iteration**
- **Policy iteration**

Thank You!

Morada: Campus de Campolide, 1070-312 Lisboa, Portugal

Tel: +351 213 828 610 | Fax: +351 213 828 611

Acreditações e Certificações da NOVA IMS

Cofinanciado por