Reinforcement Learning

Chapter 21

TB Artificial Intelligence

INSTITUT Mines-Télécom

MINES Saint-Étienne

Outline

Agents and Machine Learning (chap. 2)

Markov Decision Problems (chap. 18)

Passive Reinforcement Learning

Active Reinforcement Learning

Topic

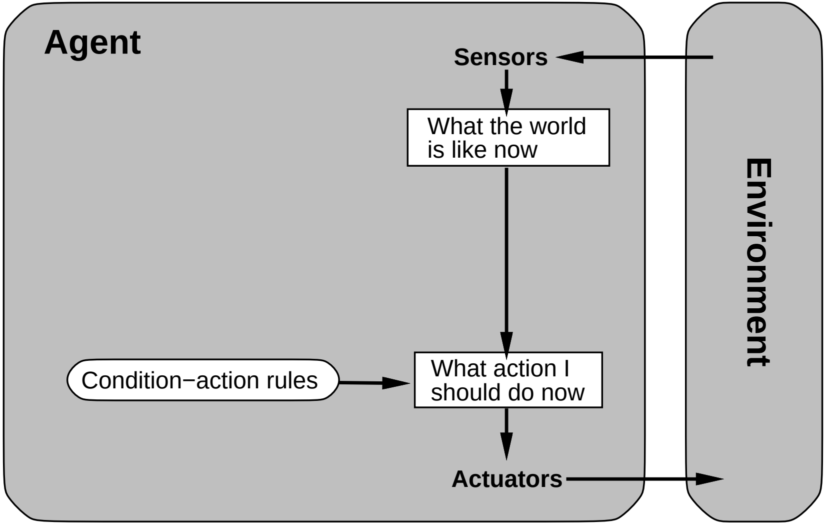
Agents and Machine Learning (chap. 2)

Markov Decision Problems (chap. 18)

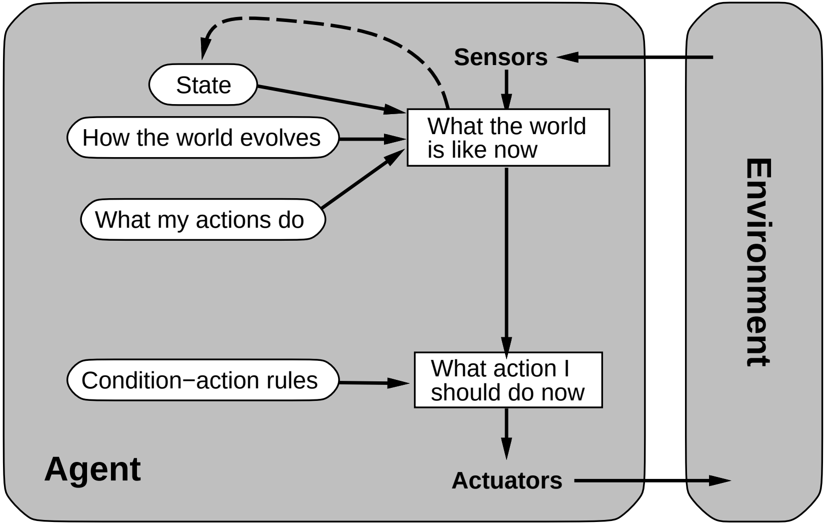
Passive Reinforcement Learning

Active Reinforcement Learning

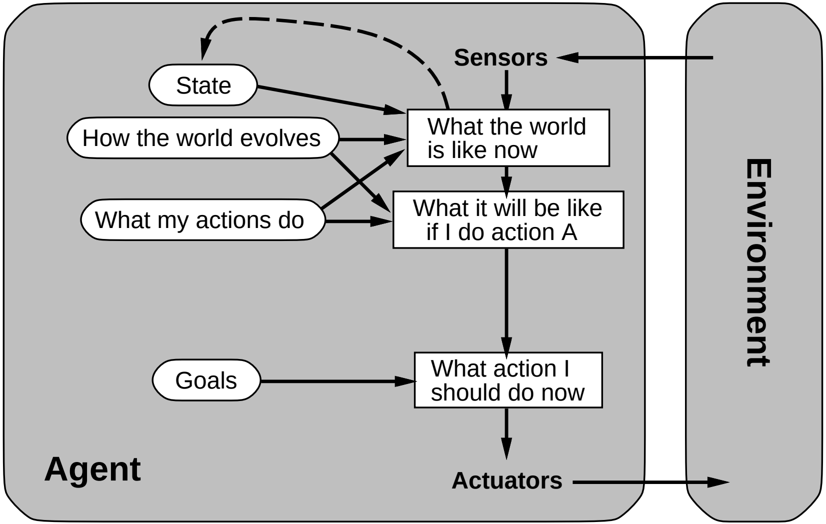
Simple Reflex Agent



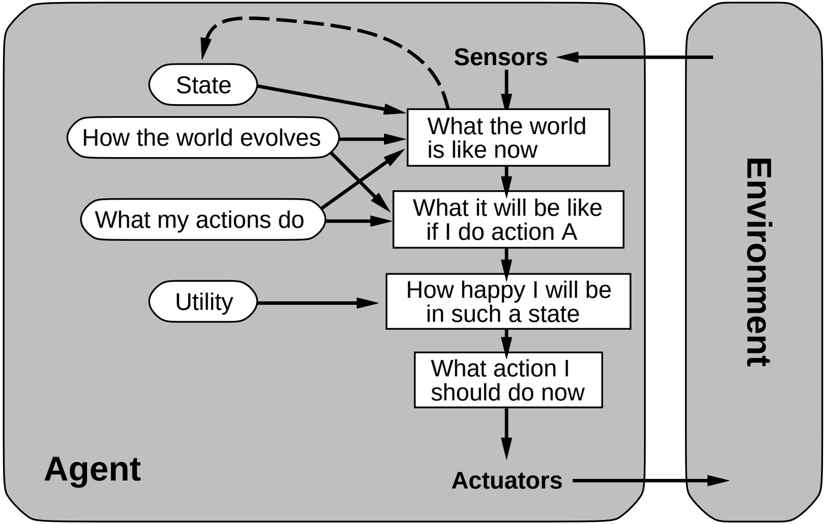
Model-based Reflex Agent



Goal-based Agent



Utility-based Agent



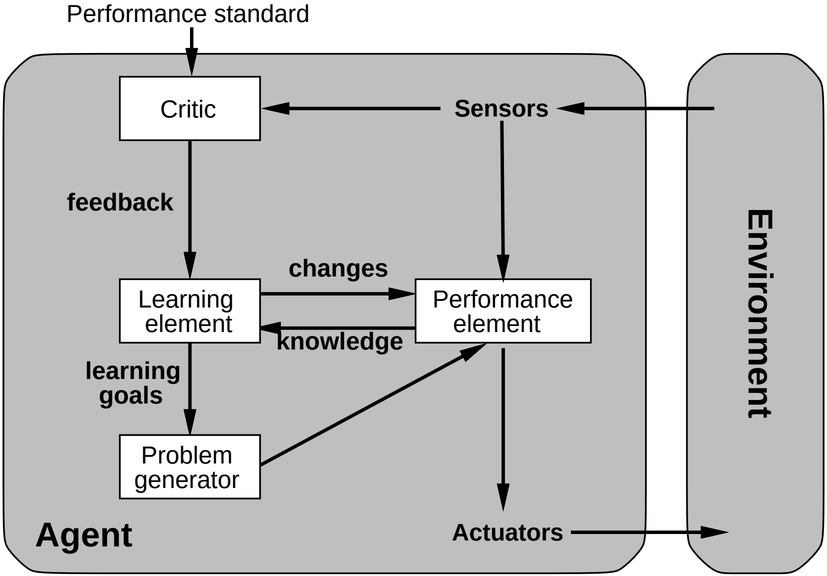
Learning Agent

4 basic kinds of intelligent agents (increasing order of complexity):

1. Simple Reflex agents
2. Model-based Reflex Agents
3. Goal-based Reflex Agents
4. Utility-based Reflex Agents

But it can be fastidious to program such agents...

Learning Agent (cont.)



Four Main Aspects

1. Which **feedback** from environnement is available?
   (supervised learning, unsupervised, reinforcement)
2. How knowledge is modeled?
   (algebraic expressions, production rules, graph, networks, sequences, . . .)
   Is there prior knowledge?
3. Which agent **components** must learn?
   - State $\rightarrow$ Action?
   - Environnement?
   - How the world evolves?
   - Predictable results of actions?
   - Desirability of actions ?
   - States which maximise utility ?
4. On-line or batch ?

Machine Learning

- Supervised Learning
  - Learning with labelled instances
  - Ex. : Decision Trees, Neural Networks, SVMS, ...
- Unsupervised Learning
  - Learning without labels
  - Ex. : K-means, clustering, ...
- Reinforcement Learning
  - Learning with rewards
  - App. : robots, autonomous vehicles, ...

Reinforcement Learning

- Supervised learning is simplest and best-studied type of learning
- Another type of learning tasks is learning behaviors when we don't have a teacher to tell us how
- The agent has a task to perform; it takes some actions in the world; at some later point gets feedback telling it how well it did on performing task
- The agent performs the same task over and over again
- The agent gets carrots for good behavior and sticks for bad behavior
- It's called reinforcement learning because the agent gets positive reinforcement for tasks done well and negative reinforcement for tasks done poorly

Reinforcement Learning (cont.)

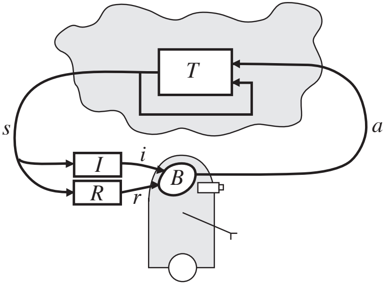
- The problem of getting an agent to act in the world so as to maximize its rewards
- Consider teaching a dog a new trick: you cannot tell it what to do, but you can reward/punish it if it does the right/wrong thing. It has to figure out what it did that made it get the reward/punishment, which is known as the credit assignment problem
- We can use a similar method to train computers to do many tasks, such as playing backgammon or chess, scheduling jobs, and controlling robot limbs

Example : SDyna for video games

Reinforcement Learning : Basic Model

[Kaelbling et al., 1996]

- i : input (some indications about s)
- s : state of the environment
- a : action
- r : reinforcement signal
- B : agent's behavior
- T : transition function
- I : input function (what is seen about the env.)



Topic

Agents and Machine Learning (chap. 2)

Markov Decision Problems (chap. 18)

Passive Reinforcement Learning

Active Reinforcement Learning

Markov Decision Process

Definition (Markov Decision Process (MDP))

A sequential decision problem for a fully observable, stochastic environment with a Markovian transition function and additive reward is called a Markov Decision Problem, and defined by a tuple $\langle S, A, T, R \rangle$:

- A set of **states** $s \in S$
- A set of **actions** $a \in A$
- A stochastic **transition function** $T(s, a, s') = P(s'|s, a)$
- A **reward** function $R(s)$

Markov Decision Process (cont.)

-0.04	-0.04	-0.04	$\boxed{+1}$
-0.04		-0.04	$\boxed{-1}$
START	-0.04	-0.04	-0.04

with probability to go straight = 0.8, left = 0.1 and right = 0.1

- In a deterministic environment, a solution would be [$\uparrow$,$\uparrow$,$\rightarrow\rightarrow\rightarrow$]
- In our stochastic environment, the probability of reaching goal state +1 given this sequence of actions is only 0.33

Markov Decision Process: Policy

- A **policy** is a function $\pi : S \rightarrow A$ that specifies what action the agent should take in any given state
- Executing a policy can give rise to many action sequences!
- How can we determine the quality of a policy?

$\rightarrow$	$\rightarrow$	$\rightarrow$	$\boxed{+1}$
$\uparrow$		$\uparrow$	$\boxed{-1}$
$\uparrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$

Markov Decision Process: Utility

- **Utility** is an internal measure of an agent's success
  - Agent's own internal **performance measure**
  - Surrogate for success and happiness
- The utility is a function of the rewards

$$
\begin{aligned}
U([s_0, s_1, s_2, \ldots]) &= \gamma^0 R(s_0) + \gamma^1 R(s_1) + \gamma^2 R(s_2) + \ldots \\
&= \sum_{t=0}^{\infty} \gamma^t R(s_t)
\end{aligned}
$$

with γ a discount factor

Markov Decision Process: Utility (cont.)

→	→	→	+1
↑		↑	-1
↑	←	←	←

Optimal Policy

0.812	0.868	0.918	+1
0.762		0.660	-1
0.705	0.655	0.611	0.388

Utilities

Value Iteration Algorithm

- Given an MDP, recursively formulate the utility of starting at state s

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a) U(s') \qquad \text{(Bellman Equation)}$$

- Suggest an iterative algorithm:

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a) U_i(s')$$

- Once we have $U(s)$ for all states s, we can construct the optimal policy:

$$\pi^*(s) = \operatorname*{argmax}_{a \in A(s)} \sum_{s'} P(s'|s,a) U(s')$$

Policy Iteration Algorithm

- **Policy evaluation** : given π_i compute U_i

$$U_i(s) = U^{\pi_i}(s) = E\left[\sum_{t=0}^{\infty} {}^t R(S_t)\right]$$

- **Policy improvement** = given U_i compute π_{i+1}

$$\pi_{i+1}(s) = \operatorname*{argmax}_{a \in A(s)} \sum_{s'} P(s'|s,a) U_i(s')$$

- Repeat until stability

Why Reinforcement Learning?

- We could find an optimal policy for an MDP if we know the transition model $P(s'|s,a)$
- **But**, an agent in an unknown environment does not know the transition model nor in advance what rewards it will get in new states
- We want the agent to learn to behave rationally in an unsupervised process

The purpose of RL is to learn the optimal policy based only on received rewards

Topic

Agents and Machine Learning (chap. 2)

Markov Decision Problems (chap. 18)

Passive Reinforcement Learning

Active Reinforcement Learning

Passive RL

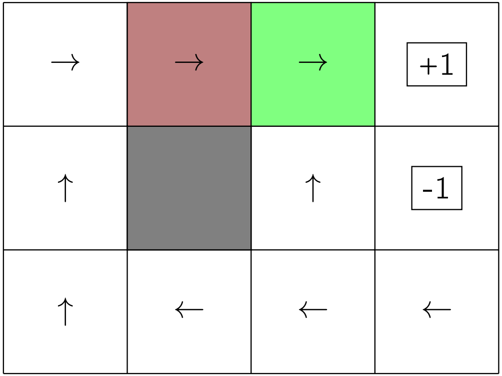
- In passive RL, the agents' policy π is fixed, it only needs to know how good it is
- Agent runs a number of trials, starting in (1,1) and continuing until it reaches a terminal state
- The utility of a state is the expected total remaining reward (reward-to-go)
- Each trial provides a sample of the reward-to-go for each visited state
- The agent keeps a running average for each state, which will converge to the true value
- This is a **direct utility estimation** method

Direct Utility Estimation

Trials

(1,1) $\uparrow$ -0.04
(1,2) $\uparrow$ -0.04
(1,3) $\rightarrow$ -0.04
(2,3) $\rightarrow$ -0.04
(3,3) $\rightarrow$ -0.04
(3,2) $\uparrow$ -0.04
(3,3) $\rightarrow$ -0.04
(4,3) exit +1
(done)

(1,1) -0.04
(1,2) $\uparrow$ -0.04
(1,3) $\rightarrow$ -0.04
(2,3) $\rightarrow$ -0.04
(3,3) $\rightarrow$ -0.04
(3,2) $\uparrow$ -0.04
(4,2) exit -1
(done)



$\gamma = 1,\ R = -0.04$

$V(2,3) \approx (0.840 - 1.12)/2 = -0.14$

$V(3,3) \approx (0.96 + 0.88 - 1.08)/3 = 0.86$

Problems with Direct Utility Estimation

- Direct utility fails to exploit the fact that states are dependent as shown in Bellman equations

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

- Learning can be speeded up by using these dependencies
- Direct utility can be seen as inductive learning search in a too large hypothesis space that contains many hypothesis violating Bellman equations

Adaptive Dynamic Programming

- An ADP agent uses dependencies between states to speed up value estimation
- It follows a policy π and can use observed transitions to incrementally built the transition model $P(s'|s, \pi(s))$
- It can then plug the learned transition model and observed rewards $R(s)$ into the Bellman equations to $U(s)$
  - The equations are linear because there is no $\max$ operator $\rightarrow$ easier to solve
- The result is $U(s)$ for the given policy π

Temporal-difference Learning

- TD is another passive utility value learning algorithm using Bellman equations
- Instead of solving the equations, TD uses the observed transitions to adjust the utilities of the observed states to agree with Bellman equations
- TD uses a **learning rate** parameter α to select the rate of change of utility adjustment
- TD does not need a trnasition model to perform its updates, only the observed transitions

Temporal-difference Learning (cont.)

- TD update rule for transition from s to s':

$$V^\pi(s) \leftarrow V^\pi(s) + \alpha(\boxed{R(s) + \gamma V^\pi(s')} - V^\pi(s))$$

(noisy) sample of value at s based on next state s'

- So the updates is maintaining a « mean » of the (noisy) value sample
- If the learning rate decreases appropriately with the number of samples (e.g. $1/n$) then the value estimates will converge to true values!

$$V^\pi(s) = R(s) + \gamma \sum_{s'} T(s, a, s') V^\pi(s')$$

(Dan Klein, UC Berkeley)

Topic

Agents and Machine Learning (chap. 2)

Markov Decision Problems (chap. 18)

Passive Reinforcement Learning

Active Reinforcement Learning

Active Reinforcement Learning

- While a passive RL agent executes a fixed policy π, an **active RL agent** has to decide which actions to take
- An active RL agent is an extension of a passive one, e.g. the passive ADP agent, and adds
  - Needs to learn a complete transition model for **all** actions (not just π), using passive ADP learning
  - Utilities need to reflect the optimal policy π^*, as expressed by the Bellman equations
  - Equations can be solved by VI or PI methods described before
  - Action to be selected as the optimal/maximizing one

(Roar Fjellheim, University of Oslo)

Exploitation vs. Exploration

- The active RL agent may select maximizing actions based on a faulty learned model, and fail to incorporate observations that might lead to a more correct model
- To avoid this, the agent design could include selecting actions that lead to more correct models at the cost of reduced immediate rewards
- This called **exploitation vs. exploration** tradeoff
- The issue of optimal exploration policy is studied in a subfield of statistical decision theory dealing with so-called bandit problems

(Roar Fjellheim, University of Oslo)

Q-Learning

- An action-utility function Q assigns an expected utility to taking a given action in a given state: $Q(a, s)$ is the value of doing action a in state s
- Q-values are related to utility values:

$$U(s) = \max_a Q(a, s)$$

- Q-values are sufficient for decision making without needing a transition model $P(s'|s, a)$
- Can be learned directly from rewards using a TD-method based on an update equation:

$$Q(s, a) \leftarrow Q(s, a) + \alpha(R(s) + \max_{a'} Q(s', a') - Q(s, a))$$

(Roar Fjellheim, University of Oslo)

Q-Learning

- Q-Learning: samplebased Q-value iteration
- Learn $Q^*(s,a)$ values
  - Receive a sample (s, a, s', r)
  - Consider your old estimate: $Q(s,a)$
  - Consider your new sample estimate:

$$Q^*(s,a) = \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q^*(s', a,') \right]$$

$$sample = R(s,a,s') + \gamma \max_{a'} Q^*(s', a,')$$

  - Incorporate the new estimate into a running average:

$$Q(s,a) \leftarrow (1-\alpha) Q(s,a) + \alpha [sample]$$

(Dan Klein, UC Berkeley)

SARSA

- Updating the Q-value depends on the current state of the agent s_1, the action the agent chooses a_1, the reward r the agent gets for choosing this action, the state s_2 that the agent will now be in after taking that action, and finally the next action a_2 the agent will choose in its new state

$$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma Q(s',a') - Q(s,a))$$

- SARSA learns the Q values associated with taking **the policy it follows itself**, while Q-learning learns the Q values associated with taking the exploitation policy while following an exploration/exploitation policy

$$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

SARSA

- Updating the Q-value depends on the current state of the agent s_1, the action the agent chooses a_1, the reward r the agent gets for choosing this action, the state s_2 that the agent will now be in after taking that action, and finally the next action a_2 the agent will choose in its new state

$$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma Q(s',a') - Q(s,a))$$

- SARSA learns the Q values associated with taking **the policy it follows itself**, while Q-learning learns the Q values associated with taking the exploitation policy while following an exploration/exploitation policy

$$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma \boxed{\max_{a'} Q(s',a')} - Q(s,a))$$

Generalization in RL

- In simple domains, U and Q can be represented by tables, indexed by state s
- However, for large state spaces the tables will be too large to be feasible, e.g. chess 10^{40} states
- Instead functional approximation can sometimes be used, e.g. $\widetilde{U}(s) = \sum parameter_i \times feature_i(s)$
- Instead of e.g. 10^{40} table entries, U can be estimated by e.g. 20 parameterized features
- Parameters can be found by supervised learning
- Problem: Such a function may not exist, and learning process may therefore fail to converge

(Roar Fjellheim, University of Oslo)

Summary

- Reinforcement learning (RL) examines how the agent can learn to act in an unknown environment just based on percepts and rewards
- Three RL designs are model-based, using a model P and utility function U, model-free, using action-utility function Q, and reflex, using a policy
- The utility of a state is the expected sum of rewards received up to the terminal state. Three methods are direct estimation, Adaptive dynamic programming (ADP), and Temporal-Difference (TD)
- Action-value function (Q-functions) can be learned by ADP or TD approaches
- In passive learning the agent just observes the environment, while an active learner must select actions to trade off immediate reward vs. exploration for improved model precision
- In domains with very large state spaces, utility tables U are replaced by approximate functions
- Policy search work directly on a representation of the policy, improving it in an iterative cycle
- Reinforcement learning is a very active research area, especially in robotics