# Lecture 16: Reinforcement learning (I)

- The reinforcement learning problem
- What to learn: policies and value functions
- Monte Carlo learning and its relationship to supervised learning



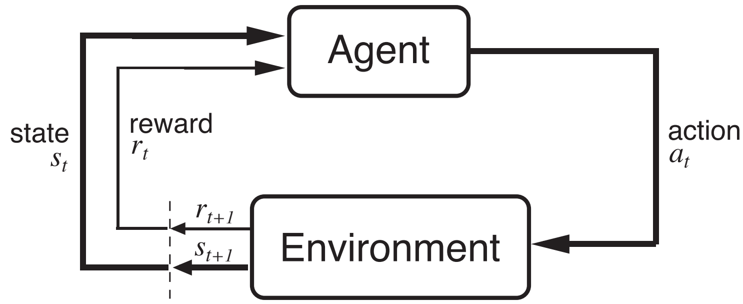
# The general problem: Control Learning

Consider *learning to choose actions*, e.g.,

- Robot learning to dock on battery charger
- Choose actions to optimize factory output
- Playing Backgammon, Go, Poker, ...
- Choosing medical tests and treatments
- Conversation
- Portofolio management
- Flying a helicopter
- Queue / router control

*All of these are sequential decision making problems*

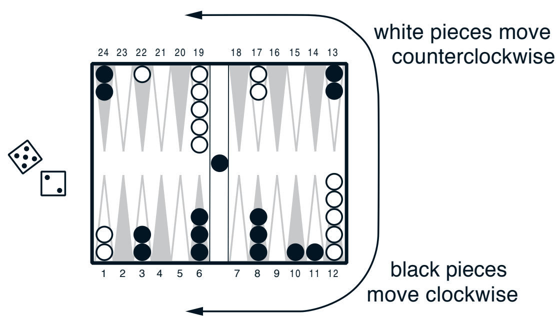
## Reinforcement Learning Problem




- At each discrete time $t$, the agent (learning system) observes state $s_t \in S$ and chooses action $a_t \in A$
- Then it receives an immediate reward $r_{t+1}$ and the state changes to $s_{t+1}$

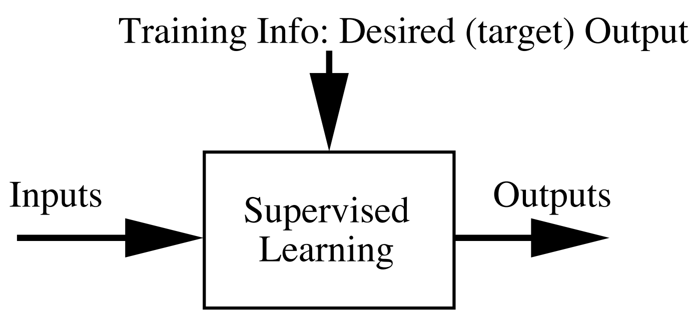


## Example: Backgammon (Tesauro, 1992-1995)




- The states are board positions in which the agent can move
- The actions are the possible moves
- Reward is $0$ until the end of the game, when it is $\pm 1$ depending on whether the agent wins or loses

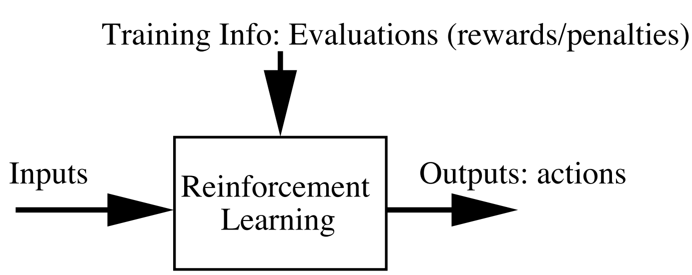
## Supervised Learning




Error = (target output - actual output)



## Reinforcement Learning (RL)



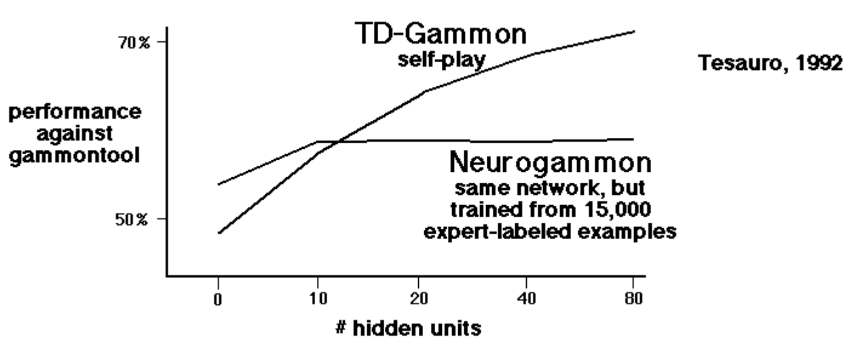

*Objective: Get as much reward as possible*

## Key Features of RL

- The learner is not told what actions to take, instead it find finds out what to do by *trial-and-error search*
- The environment is *stochastic*
- The *reward may be delayed*, so the learner may need to sacrifice short-term gains for greater long-term gains
- The learner had to balance the need to *explore* its environment and the need to *exploit* its current knowledge



## The Power of Learning from Experience




- Expert examples are expensive and scarce
- Experience is cheap and plentiful!

## Agent's Learning Task

Execute actions in environment, observe results, and learn **policy** (strategy, way of behaving) $\pi : S \times A \rightarrow [0,1]$,

$$\pi(s,a) = P\left(a_t = a | s_t = s\right)$$

If the policy is deterministic, we will write it more simply as $\pi : S \rightarrow A$, with $\pi(s) = a$ giving the action chosen in state $s$.

- Note that the target function is $\pi : S \rightarrow A$ but we have *no training examples* of form $\langle s, a \rangle$
  Training examples are of form $\langle \langle s, a \rangle, r, s', \dots \rangle$
- Reinforcement learning methods specify how the agent should change the policy $\pi$ as a function of the rewards received over time



## The objective: Maximize long-term return

Suppose the sequence of rewards received after time step $t$ is $r_{t+1}, r_{t+2} \dots$. We want to maximize the **expected return** $E[R_t]$ for every time step $t$

- *Episodic tasks*: the interaction with the environment takes place in episodes (e.g. games, trips through a maze etc)

$$R_t = r_{t+1} + r_{t+2} + \dots + r_T$$

where $T$ is the time when a terminal state is reached

## The objective: Maximize long-term return

Suppose the sequence of rewards received after time step $t$ is $r_{t+1}, r_{t+2} \ldots$. We want to maximize the **expected return** $E\{R_t\}$ for every time step $t$

- *Discounted continuing tasks* :

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=1}^{\infty} \gamma^{t+k-1} r_{t+k}$$

where $\gamma$ = discount factor for later rewards (between 0 and 1, usually close to 1)

Sometimes viewed as an "inflation rate" or "probability of dying"



## The objective: Maximize long-term return

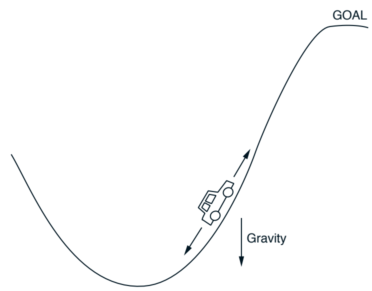
Suppose the sequence of rewards received after time step $t$ is $r_{t+1}, r_{t+2} \ldots$. We want to maximize the **expected return** $E\{R_t\}$ for every time step $t$

- *Average-reward tasks*:

$$R_t = \lim_{T \to \infty} \frac{1}{T} (r_{t+1} + r_{t+2} + \cdots + r_T)$$

This represents the reward per time step.

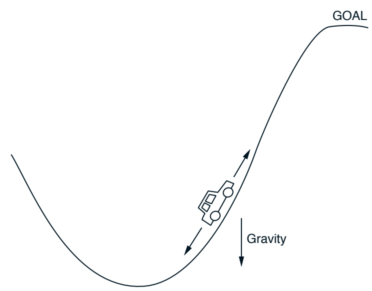
## Example: Mountain-Car




- States: position and velocity
- Actions: accelerate forward, accelerate backward, coast
- We want the car to get to the top of the hill as quickly as possible
- What are the rewards and the return?

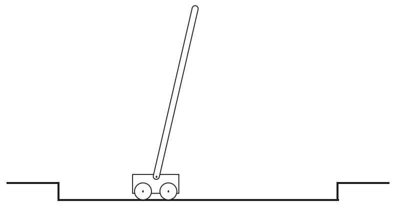


## Example: Mountain-Car




- States: position and velocity
- Actions: accelerate forward, accelerate backward, coast
- Two reward formulations:
  - reward = $-1$ for every time step, until car reaches the top
  - reward = $1$ at the top, 0 otherwise $\gamma < 1$
- In both cases, the return is maximized by minimizing the number of steps to the top of the hill

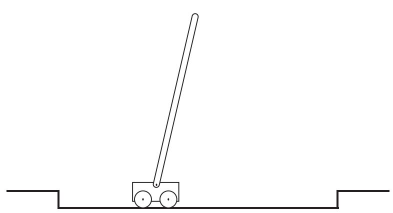
## Example: Pole Balancing



Avoid failure: pole falling beyond a given angle, or cart hitting the end of the track



## Example: Pole Balancing



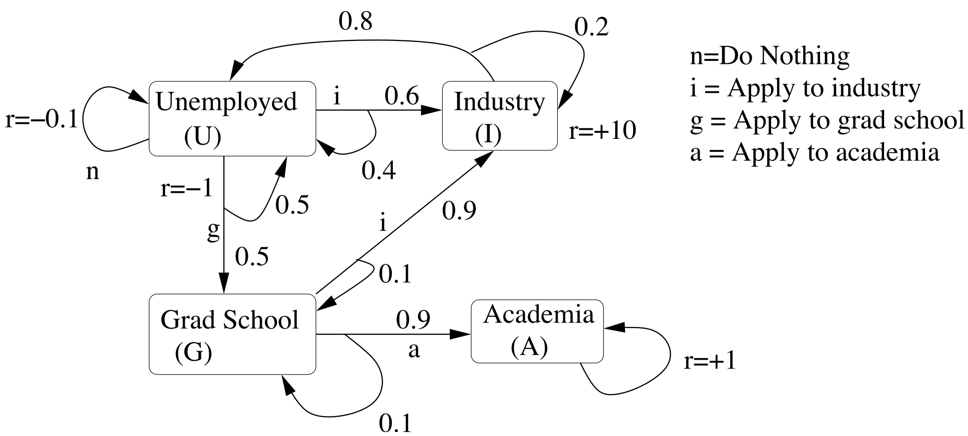
Avoid failure: pole falling beyond a given angle, or cart hitting the end of the track

- Episodic task formulation: reward = +1 for each step before failure

  $\Rightarrow$ return = number of steps before failure
- Continuing task formulation: reward = -1 upon failure, 0 otherwise, $\gamma < 1$

  $\Rightarrow$ return = $-\gamma^k$ if there are $k$ steps before failure

## Graduate school example




What is the best policy?



## Finding a good policy

- The problem seems difficult to solve even for toy examples
- Since we do not have expert-labeled examples, ideas for supervised learning do not apply immediately.
- One way to address the problem is to use *search for a good policy*, in the space of all possible policies
- To do this, we need a *measure of the quality* of a policy

## State Value Function

- The **value of a state $s$** under policy $\pi$ is the expected return when starting from $s$ and choosing actions according to $\pi$:

$$V^{\pi}(s) = E_{\pi}\{R_0 \mid s_0 = s\} = E_{\pi}\left\{\sum_{k=1}^{\infty} \gamma^{k-1} r_k \mid s_0 = s\right\}$$

- If the state space is finite, the collection of values of all states, $V^{\pi}$, can be represented as a vector of size equal to the number of states.
- This vector is called the **state-value function**



## State-action value function

- Analogously, the **value of taking action $a$ in state $s$** under policy $\pi$ is:

$$Q^{\pi}(s, a) = E_{\pi}\left\{\sum_{k=1}^{\infty} \gamma^{k-1} r_k \mid s_0 = s, a_0 = a\right\}$$

- $Q^{\pi}$ can be represented as a matrix of size $|S| \times |A|$; this is called the **action-value function**

## Policies and value functions

- Value functions define a *partial order over policies*:

$$\pi_1 \geq \pi_2 \text{ if and only if } V^{\pi_1}(s) \geq V^{\pi_2}(s) \forall s \in S$$

- So a policy is "better" than another policy if and only if it generates at least the same amount of return *at all states*
- If $\pi_1$ has higher value than $\pi_2$ at some states and lower value at other, the two policies are not comparable.
- Computing the value of a policy will be helpful in searching for it.



## Monte Carlo Methods

- Suppose we have an episodic task
- The agent behaves according to some policy $\pi$ for a while, generating several trajectories.
- Compute $V^\pi(s)$ by *averaging the observed returns* after $s$ on the trajectories in which $s$ was visited.
- Two main approaches:
  - **Every-visit:** average returns for every time a state is visited in an episode
  - **First-visit:** average returns only for the first time a state is visited in an episode

## Implementation of Monte Carlo Policy Evaluation

Suppose that we have $n+1$ returns from state $s$

$$
\begin{aligned}
V^{n+1}(s) &= \frac{1}{n+1}\sum_{i=1}^{n+1} R^i(s) = \frac{1}{n+1}\left(\sum_{i=1}^{n} R^i(s) + R^{n+1}(s)\right) \\
&= \frac{n}{n+1}\frac{1}{n}\sum_{i=1}^{n} R^i(s) + \frac{1}{n+1}R^{n+1}(s) \\
&= \frac{n}{n+1}V^n(s) + \frac{1}{n+1}R^{n+1}(s) \\
&= V^n(s) + \frac{1}{n+1}\left(R^{n+1}(s) - V^n(s)\right)
\end{aligned}
$$

If we do not want to keep counts of how many times states have been visited, we can use a *learning rate* version:

$$V(s_t) \leftarrow V(s_t) + \alpha_t(R_t - V(s_t))$$



## Monte Carlo estimation of action values

- We use the same idea: $Q^\pi(s,a)$ is the average of the returns obtained by starting in state $s$, doing action $a$ and then choosing actions according to $\pi$
- Like the state-value version, it converges asymptotically *if every state-action pair is visited*
- But $\pi$ might not choose every action in every state!
- **Exploring starts:** Every state-action pair has a non-zero probability of being the starting pair

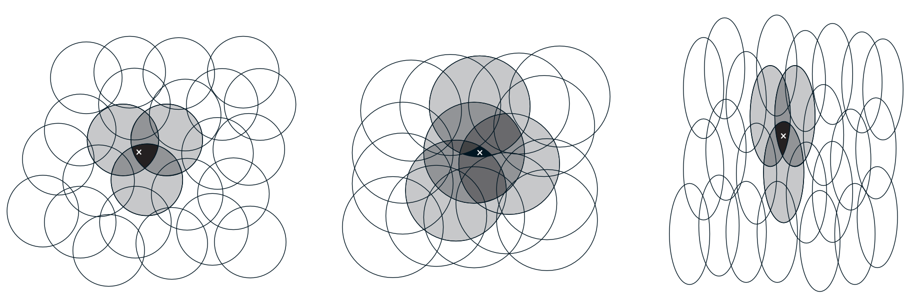
## Representing value functions

- If the state space is finite, $V^\pi$ can be represented as an array with one entry for every state
- If the state space is infinite, use your favorite function approximator that can represent real-values functions:
  - Linear function approximator, with non-linear basis functions
  - Nearest neighbor
  - Neural networks
  - Locally weighted regression
  - Regression trees
  - ...
- Some choices are better than others, theoretically and in practice.



## Sparse, coarse coding

Main idea: we want linear function approximators (because they have good convergence guarantees, as we will see later) but with **lots of features**, so they can represent complex functions



a) Narrow generalization b) Broad generalization c) Asymmetric generalization

- *Coarse* means that the receptive fields are typically large
- *Sparse* means that just a few units are active ar any given time

E.g., CMACs, sparse distributed memories etc.