Fundamentals of Reinforcement Learning

December 9, 2013 - Techniques of AI

Yann-Michaël De Hauwere - ydehauwe@vub.ac.be

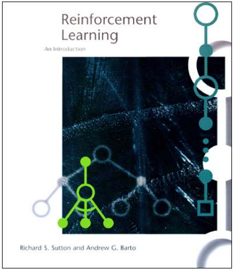
Course material

Slides online

T. Mitchell
Machine Learning, chapter 13
McGraw Hill, 1997

Richard S. Sutton and Andrew G. Barto
Reinforcement Learning: An Introduction
MIT Press, 1998

Available on-line for free!



Why reinforcement learning?

Based on ideas from psychology

- Edward Thorndike's **law of effect**
  - Satisfaction strengthens behavior, discomfort weakens it
- B.F. Skinner's **principle of reinforcement**
  - Skinner Box: train animals by providing (positive) feedback

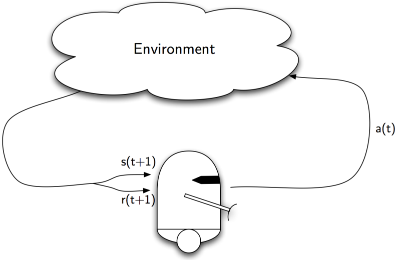
Learning by interacting with the environment

Why reinforcement learning?

Control learning

- Robot learning to dock on battery charger
- Learning to choose actions to optimize factory output
- Learning to play Backgammon/other games

The RL setting



- Learning from interactions
- Learning what to do - **how to map situations to actions** - so as to maximize a numerical reward signal

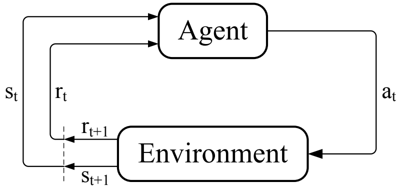
Key features of RL

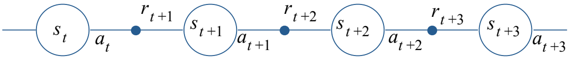
- Learner is **not** told which action to take
- Trial-and-error approach
- Possibility of **delayed reward**
  - Sacrifice short-term gains for greater long-term gains
- Need to balance **exploration** and **exploitation**
- Possible that states are only partially observable
- Possible needs to learn multiple tasks with same sensors
- In between **supervised** and **unsupervised** learning

The agent-environment interface

Agent interacts at discrete time steps $t = 0, 1, 2, \ldots$

- Observes state $s_t \in S$
- Selects action $a_t \in A(s_t)$
- Obtains immediate reward $r_{t+1} \in \mathfrak{R}$
- Observes resulting state s_{t+1}





Elements of RL

- Time steps need not refer to fixed intervals of real time
- **Actions** can be
  - low level (voltage to motors)
  - high level (go left, go right)
  - "mental" (shift focus of attention)
- **States** can be
  - low level "sensations" (temperature, (x, y) coordinates)
  - high level abstractions, symbolic
  - subjective, internal ("surprised", "lost")
- The **environment** is not necessarily known to the agent

Elements of RL

- **State transitions** are
  - changes to the internal state of the agent
  - changes in the environment as a result of the agent's action
  - can be nondeterministic
- **Rewards** are
  - goals, subgoals
  - duration
  - ...

Learning how to behave

- The agent's **policy** π at time t is
  - a mapping from states to action probabilities
  - $\pi_t(s, a) = P(a_t = a | s_t = s)$

- Reinforcement learning methods specify **how** the agent changes its policy as a result of experience
- Roughly, the agent's goal is to get **as much reward** as it can over the long run

The objective

- Use **discounted return** instead of total reward

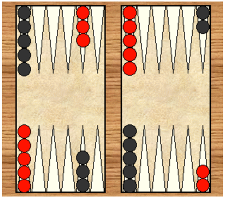
$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

where $\gamma \in [0, 1]$ is the **discount factor** such that

shortsighted $\quad 0 \leftarrow \gamma \rightarrow 1 \quad$ farsighted

Example: backgammon

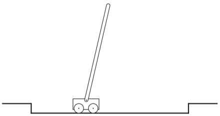
- Learn to play backgammon
- Immediate reward:
  - $+100$ if win
  - -100 if lose
  - 0 for all other states



Trained by playing 1.5 million games against itself
Now approximately equal to best human player.

Example: pole balancing

- A **continuing** task with discounted return:
  - reward = -1 upon failure
  - return $= -\gamma^k$, for k steps before failure



Return is maximized by avoiding failure for as long as possible

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

Examples: pole balancing (movie)

Markov decision processes

- It is often useful to a assume that all relevant information is present in the current state: **Markov property**

$$P(s_{t+1}, r_{t+1}|s_t, a_t) = P(s_{t+1}, r_{t+1}|s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0)$$

- If a reinforcement learning task has the Markov property, it is basically a **Markov Decision Process (MDP)**
- Assuming finite state and action spaces, it is a finite MDP

Markov decision processes

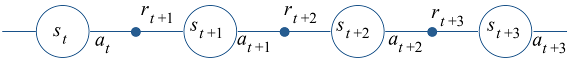
An MDP is defined by

- **State and action sets**
- a **Transition function**

$$\mathcal{P}^{a}_{ss'} = P(s_{t+1} = s' | s_t = s, a_t = a)$$

- a **Reward function**

$$\mathcal{R}^{a}_{ss'} = E(r_{t+1} | s_t = s, a_t = a, s_{t+1} = s')$$



Value functions

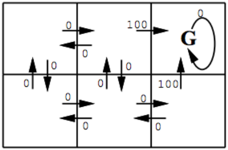
- Goal: learn $\pi : S \rightarrow A$, given $\langle\langle s, a\rangle, r\rangle$
- When following a fixed policy π we can define the **value** of a state s under that policy as

$$V^{\pi}(s) = E_{\pi}(R_t | s_t = s) = E_{\pi}(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s)$$

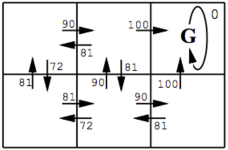
- Similarly we can define the value of taking action a in state s as

$$Q^{\pi}(s, a) = E_{\pi}(R_t | s_t = s, a_t = a)$$

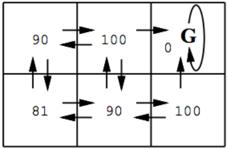
- Optimal $\pi^* = argmax_{\pi} V^{\pi}(s)$



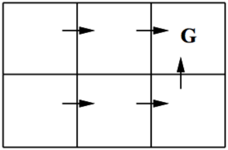
$r(s,a)$ (immediate reward) values



$Q(s,a)$ values



$V^*(s)$ values



One optimal policy

Value functions

- The value function has a particular recursive relationship, expressed by the **Bellman equation**

$$V^\pi(s) = \sum_{a \in A(s)} \pi(s,a) \sum_{s' \in S} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V^\pi(s')]$$

- The equation expresses the recursive relation between the value of a state and its successor states, and averages over all possibilities, weighting each by its probability of occurring

Learning an optimal policy online

- Often transition and reward functions are unknown
- Using **temporal difference (TD)** methods is one way of overcoming this problem
  - Learn directly from raw experience
  - No model of the environment required (model-free)
  - E.g.: **Q-learning**
- Update predicted state values based on new observations of immediate rewards and successor states

Q-function

$$Q(s,a) = r(s,a) + \gamma V^*(\delta(s,a))\text{with } s_{t+1} = \delta(s_t, a_t)$$

- if we know Q, we do not have to know δ.

$$\pi^*(s) = argmax_a[r(s,a) + \gamma V^*(\delta(s,a))]$$

$$\pi^*(s) = argmax_a Q(s,a)$$

Training rule to learn Q

- Q and V^* are closely related:

$$V^*(s) = max_{a'} Q(s, a')$$

- which allows us to write Q as:

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma V^*(\delta(s_t, a_t))$$

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma max_{a'} Q(s_{t+1}, a')$$

- So if $\hat{Q}$ represents the learner's current approximation of Q:

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

Q-learning

- Q-learning updates state-action values based on the immediate reward and the optimal expected return

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

- Directly learns the optimal value function independent of the policy being followed
- Proven to converge to the optimal policy given "sufficient" updates for each state-action pair, and decreasing learning rate α [Watkins92,Tsitsiklis94]

Q-learning

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
    Initialize s
    Repeat (for each step of episode):
        Choose a from s using policy derived from Q (e.g., ε-greedy)
        Take action a, observe r, s'
        $Q(s, a) \leftarrow Q(s, a) + \alpha\big[r + \gamma \max_{a'} Q(s', a') - Q(s, a)\big]$
        $s \leftarrow s'$;
    until s is terminal

Action selection

- How to select an action based on the values of the states or state-action pairs?
- Success of RL depends on a **trade-off**
  - Exploration
  - Exploitation
- **Exploration** is needed to prevent getting stuck in local optima
- To ensure convergence you need to **exploit**

Action selection

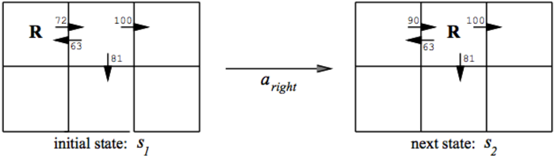
Two common choices

- **ϵ-greedy**
  - Choose the best action with probability $1 - \epsilon$
  - Choose a random action with probability ϵ
- **Boltzmann exploration** (softmax) uses a temperature parameter τ to balance exploration and exploitation

$$\pi_t(s, a) = \frac{e^{Q_t(s,a)/\tau}}{\sum_{a' \in A} e^{Q_t(s,a')/\tau}}$$

pure exploitation $\quad 0 \leftarrow \tau \rightarrow \infty \quad$ pure exploration

Updating Q: in practice



$$\begin{aligned}\hat{Q}(s_1, a_{right}) &\leftarrow r + \gamma \max_{a'} \hat{Q}(s_2, a') \\ &\leftarrow 0 + 0.9 \max\{63, 81, 100\} \\ &\leftarrow 90\end{aligned}$$

notice if rewards non-negative, then

$$(\forall s, a, n) \;\; \hat{Q}_{n+1}(s, a) \geq \hat{Q}_n(s, a)$$

and

$$(\forall s, a, n) \;\; 0 \leq \hat{Q}_n(s, a) \leq Q(s, a)$$

Convergence of deterministic Q-learning

$\hat{Q}$ converges to Q when each $\langle s, a \rangle$ is visited infinitely often

Proof:

- Let a full interval be an interval during which each $\langle s, a \rangle$ is visited
- Let $\hat{Q}_n$ be the Q-table after n-updates
- Δ_n is the maximum error in $\hat{Q}_n$:

$$\Delta_n = max_{s,a}|\hat{Q}_n(s,a) - Q(s,a)|$$

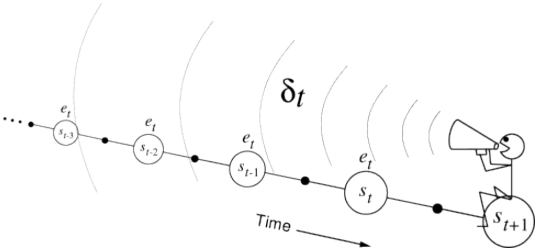
Convergence of deterministic Q-learning

For any table entry $\hat{Q}_n(s,a)$ updated on iteration $n+1$, the error in the revised estimate is $\hat{Q}_{n+1}(s,a)$

$$\begin{aligned}
\hat{Q}_{n+1}(s,a) - Q(s,a)| &= |(r + \gamma max_{a'} \hat{Q}_n(s',a')) \\
&\quad -(r + \gamma max_{a'} Q(s',a'))| \\
&= |\gamma max_{a'} \hat{Q}_n(s',a')) - \gamma max_{a'} Q(s',a'))| \\
&\leq \gamma max_{a'} |\hat{Q}_n(s',a') - Q(s',a'))| \\
&\leq \gamma max_{s'',a'} |\hat{Q}_n(s'',a') - Q(s'',a'))| \\
\hat{Q}_{n+1}(s,a) - Q(s,a)| &\leq \gamma \Delta_n < \Delta_n
\end{aligned}$$

Extensions

- **Multi-step TD**
  - Instead of observing one immediate reward, use n consecutive rewards for the value update
  - Intuition: your current choice of action may have implications for the future
- **Eligibility traces**
  - State-action pairs are eligible for future rewards, with more recent states getting more credit



Extensions

- **Reward shaping**
  - Incorporate domain knowledge to provide additional rewards during an episode
  - Guide the agent to learn faster
  - (Optimal) policies preserved given a potential-based shaping function [Ng99]
- **Function approximation**
  - So far we have used a tabular notation for value functions
  - For large state and actions spaces this approach becomes intractable
  - Function approximators can be used to generalize over large or even continuous state and action spaces

Demo

http://wilma.vub.ac.be:3000

Questions?