Introduction to Computational Intelligence

Short introduction to reinforcement learning

Igor Farkaš
Centre for Cognitive Science
Faculty of Mathematics, Physics and Informatics
Comenius University in Bratislava

Based on Russell & Norvig: Artificial Intelligence: A Modern Approach (3rd ed). Prentice Hall, 2010.

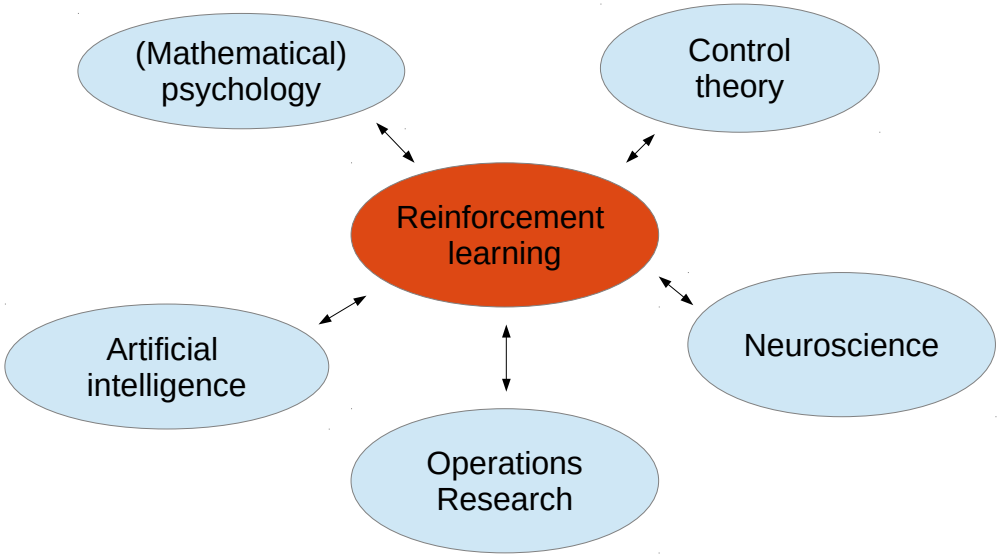
Types of machine learning

Supervised learning – error correction;
Tasks: classification, regression, prediction

Unsupervised learning – statistical correlations
Tasks: dimensionality reduction, density estimation, data visualization

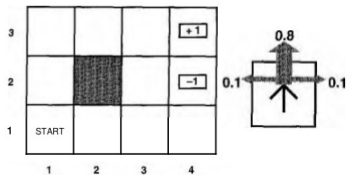
Reinforcement learning – maximizing long-term reward
Tasks: sequential decision making via interaction with the environment.

Links to other research areas



Sequential decision problems

- Agent's utility (function) depends on a sequence of decisions
- Environment assumed to be
  - discrete and finite
  - fully observable and stochastic
- agent can execute (discrete) actions in each state
- Transition function – the outcome of each action in each state
- Reward function – for agent in each state
- then we have Markov decision process
- RL works with MDP assumption
- Solution = policy π (actions taken)



Maze: 11 states, 4 actions

Markov decision process

$Sequence: s_1, a_1, r_2, s_2, .., s_t, a_t, r_{t+1}, s_{t+1}, ...$

Markov property assumed:

$$P(s_{t+1}|s_t, a_t) = P(s_{t+1}|s_t, a_t, ..., s_1, a_1)$$

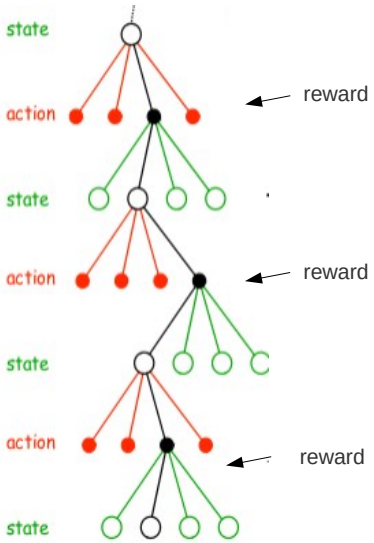
Transition function:

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

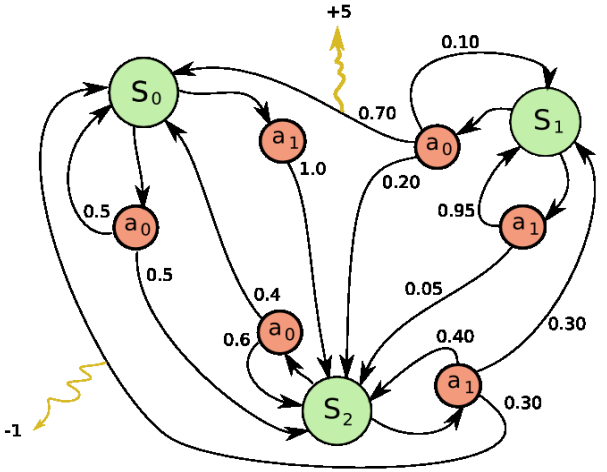
Reward function:

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

- Hence, the world is stochastic

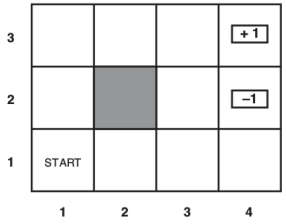


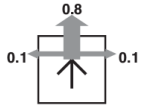
Example of MDP transition automaton



- states (S_i), 2 actions (a_j), numbers at links show $P(s'|a,s)$, and $R(s'|a,s)$
- What is the optimal policy?

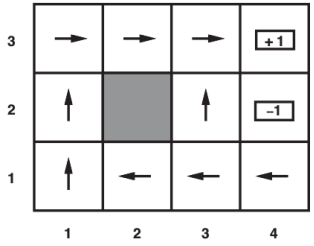
Maze example



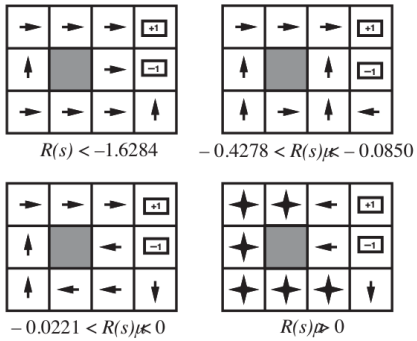


- Stochastic, fully observable environment
- State = agent's position (known)
- Probabilities of actions: 0.8, 0.1, 0.1
- What's the optimal solution (policy)?
- It depends on reward

Optimal policies:

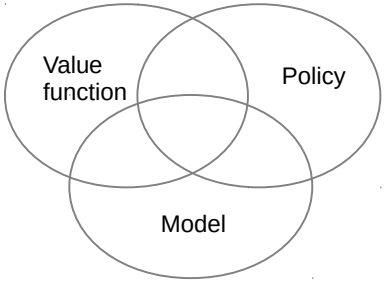


Reward at each transition: $R(s) = -0.04$



Components of a RL agent

- An RL agent may include one of more of these components:
  - Value (utility) function: how good is each state or action
  - Policy: how to behave: state → action(s)
  - Model: agent's representation of the environment
- According to its components, we get a taxonomy of RL agents:

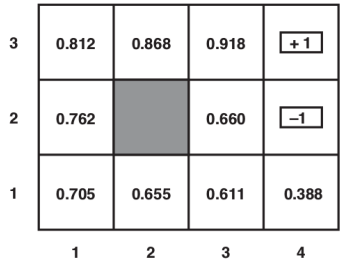


Reward, utility function and policy

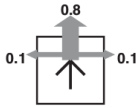
- utility (value) function – allows to choose actions
- Finite or infinite horizon for decision making?
- Additive / discounted rewards: $R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots$
- $\gamma \in (0,1\rangle$ – discount factor (future rewards are valued less)
- Utility function: $U^{\pi}(s) = E_{\pi}\{R_t | s_t = s\} = E_{\pi}\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\}$
- Optimal policy: based on $U^*(s) = max_{\pi} U^{\pi}(s)$
- Bellman equation: $U^{\pi}(s) = r(s) + \gamma \, max_a \sum_{s'} P(s'|a,s) U(s')$ (value iteration)
- In model-based RL: *Transition & Reward* functions are known

Value iteration – utilities of states

Calculate the optimal policy with Bellman eq. (iterative process): then use the utilities of states to select an optimal action in each state.



Calculated for $R(s) = -0.04$ and $\gamma = 1$



U(1,1) = – 0.04 + max {0.8*U(1,2) + 0.1*U(2,1) + 0.1*U(1,1), Up
0.9*U(1,1) + 0.1*U(1,2), Left
0.9*U(1,1) + 0.1*U(2,1), Down
0.8*U(2,1) + 0.1*U(1,2) + 0.1*U(1,1) } Right

Passive RL

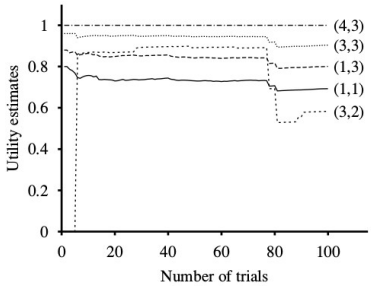
- Agent's policy is fixed (in state *s*, action $\pi(s)$ is performed)
- Goal: to learn how good the policy is (i.e. learn $U(s)$)
- passive RL agent does not know $T(s,a,s')$, nor reward-to-go $R(s)$ for each state (i.e. average reward accumulated from that state)
- Ways to learn the utility function:
  - Direct utility estimation (reward-to-go known)
  - Adaptive dynamic programming (learns the transition model in fully observable environment)
  - Temporal difference learning (model-free, approximation to ADP)

Direct utility estimation

- Estimated reward-to-go for each state
- Idea: calculate average reward over all trials (episodes) for each state independently, use as teaching signal
- Ignores connections between successive states (used in Bellman eq.)
- Hence, it converges very slowly

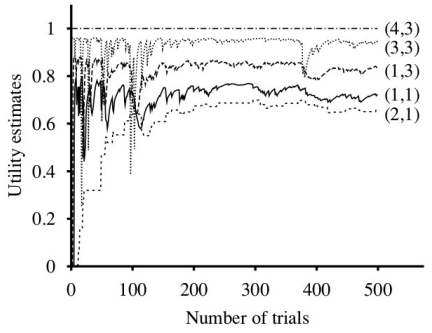
Adaptive dynamic programming

- Agent takes advantage of the constraints (resulting from links between states)
- Learns the transition model *T*(*s,a,s'*) - by estimating *P*(*s'*|*a,s*)
  - easy because environment is fully observable
  - a supervised learning task
- *P*(.) are plugged (together with obtained reward) into Bellman eq.
- But: intractable for large spaces



Temporal-difference learning

- Successive states considered, world model not learned
- Temporal difference error: $\delta_t = r_{t+1} + \gamma U_t(s_{t+1}) - U_t(s_t)$
- TD update rule: $U_{t+1}(s_t) = U_t(s_t) + \alpha \delta_t$, α - learning rate
- prediction task



Partially observable MDPs

- Agent does not know true state (due to limited sensors)
- Optimal action depends on agent's current belief state
- Agent also has a sensor (evidence) model: *P*(*e*|*s*)
- Belief state = probability distribution over the states

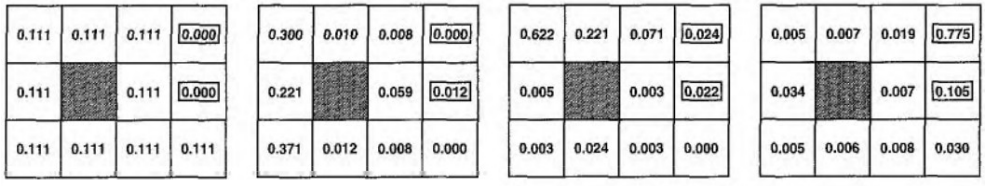


Figure 17.8 (a) The initial probability distribution for the agent's location. (b) After moving *Left* five times. (c) After moving Up five times. (d) After moving Right five times.

Review of important concepts in RL

- (numerical) reward (includes also punishment)
- RL = temporal credit assignment problem (learning from delayed reward)
- Policy – strategy for choosing actions in various states
- Exploration vs exploitation
- Model-based (we know transition function and reward function) vs model-free approaches
- Two related problems:
  - Prediction (passive RL): learning the value function for the policy followed
  - Control (active RL): learning the optimal policy (includes prediction)

Active RL – control task

- exploration enabled (non-greedy behavior), model-free
  - random factor – no guarantee
- Control task – choosing best actions $U(s) = max_a Q(s,a)$
- Learning state-action *Q*-values:
- Optimal policy: $Q^*(s,a) = max_\pi Q^\pi(s,a)$
- Q-learning update (off-policy): converges faster

$$Q_{t+1}(s_t,a_t) = Q_t(s_t,a_t) + \alpha[r_{t+1} + \gamma\, max_a Q_t(s_{t+1},a) - Q_t(s_t,a_t)]$$

- SARSA update: (on-policy)

$$Q_{t+1}(s_t,a_t) = Q_t(s_t,a_t) + \alpha[r_{t+1} + \gamma\, Q_t(s_{t+1},a_{t+1}) - Q_t(s_t,a_t)]$$

Actor–critic architectures

- Agent is split into two components
- Critic – learns the state values *V*(*s*)
- Actor – learns the policy $\pi(s)$
- Critic should not learn too quickly, nor too slowly
- A-C approach has advantages in high-dim. spaces and continuous actions (over Q-learning and SARSA)
- biological relevance

Generalization in RL

- MDP in continuous state space (and action space)
- Use function approximation
- Learning = setting parameters of this function
- Relationship to supervised learning (gradient-based methods)
- Very useful in various continuous, high-dimensional, partially observable environments (e.g. robotics).
- E.g. using actor-critic architecture (both the policy and value function are stored)

Function approximation

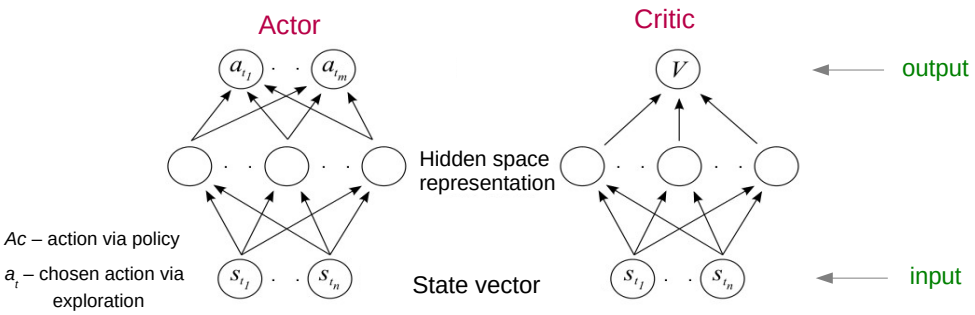
- Linear – combination of features
  - Simple to implement, faster to compute, convergence
  - Common method to find features: discretization of state space; fuzzy sets can be used (designed by hand)
- Non-linear – more general and complex
  - Less convergence guarantees
- Updating parameters of the function

Updating parameters

- In some cases, closed-form solution can be found
- Otherwise, iterative methods must be used (Bayesian methods possible for stationary functions)
- Gradient descent methods
- Gradient-free optimization – useful when the function to be optimized is not differentiable
  - evolutionary strategies can be used

Continuous A-C Learning Automaton

(van Hasselt & Wiering, 2007)



for $t = 0,1,2,..$ **do**
  choose action $a_t \leftarrow Ac_t(s_t)$, using exploration
  make action a_t, get to next state s_{t+1}
  update critic: $V_{t+1} \leftarrow r_{t+1} + \gamma V_t(s_{t+1})$
  if $V_{t+1}(s_t) > V_t(s_t)$, **then**
    update actor's parameters such that: $Ac_{t+1} \leftarrow a_t$
  end if
end for

$$\theta^V_{i,t+1} = \theta^V_{i,t} + \alpha\delta_t \frac{\partial V_t(s_t)}{\partial \theta^V_{i,t}}$$

$$\theta^{\mathrm{Ac}}_{i,t+1} = \theta^{\mathrm{Ac}}_{i,t} + \alpha(a_t - \mathrm{Ac}_t(s_t)) \frac{\partial \mathrm{Ac}_t(s_t)}{\partial \theta^{\mathrm{Ac}}_{i,t}}$$

Challenges and extensions to RL

- Curse of dimensionality
- Temporal credit assignment problem
- Partial observability problem
- State-action space tiling
- Non-stationary environments
- Credit structuring problem (origin of rewards)
- Exploration-exploitation dilemma

- Typical application areas: control, gaming

Summary

- RL = how an agent can behave in unknown environment, given only its percepts and occasional rewards
- The overall agent design dictates the kind of information that must be learned
- To be learned: utilities, optimal policy
- For large spaces – function approximation inevitable
- Great promise for robotics
- Biological relevance
- Intrinsic motivation research in RL (e.g. curiosity)
  - Active exploration strategies: goals or actions