Reinforcement learning (II)

AI 2025/2026

Content

Introduction

Passive learning
- Model-based learning: Adaptive dynamic programming
- Model-free learning: Direct utility estimation
- Model-free learning: Temporal-Difference learning

Active learning
- Q-learning
- Deep Q-learning

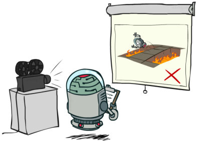
Conclusions

- Markov decision process (MDP)
  - The set of states S, the set of actions A
  - The transition model $P(s'|s,a)$ is known
  - The reward function $R(s)$ is known
  - Computes an optimal policy

- Reinforcement learning
  - It is based on MDPs, but:
  - The transition model is unknown
  - The reward function is unknown
  - Learn an optimal policy

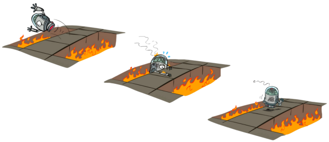
Types of reinforcement learning

- Passive: the agent learns the utility of being in certain states. The agent executes a fixed policy and evaluates it.



The agent has no control over his actions; applications: robotics.

- Active: the agent must learn what to do.



The agent must explore the environment and use the learned information. The agent updates its policy as it learns.

Types of reinforcement learning

- Model-based: learn the model of transitions and rewards and use it to discover the optimal policy
- Model-free: discover the optimal policy without learning the model

Content

Introduction

Passive learning
- Model-based learning: Adaptive dynamic programming
- Model-free learning: Direct utility estimation
- Model-free learning: Temporal-Difference learning

Active learning
- Q-learning
- Deep Q-learning

Conclusions

Passive learning

- The policy is fixed: always execute action $\pi(s)$ in state s
- The goal: learn how good is the policy π
  - learn the utility $U^{\pi}(s)$ of each state

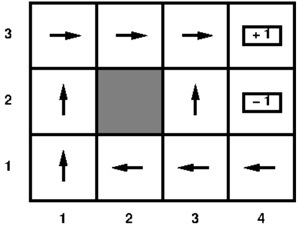
    how?

    execute the policy and learn from experience

  - similar approach to step (1) policy evaluation from the *Policy iteration* algorithm; the difference: doesn't know the transition model $P(s'|s,a)$ and $R(s)$

Passive learning is a way of exploring the environment.

Passive learning



- The agent performs a series of trials (episodes)
  $(1,1)_{-.04} \rightsquigarrow (1,2)_{-.04} \rightsquigarrow (1,3)_{-.04} \rightsquigarrow (1,2)_{-.04} \rightsquigarrow (1,3)_{-.04} \rightsquigarrow (2,3)_{-.04} \rightsquigarrow (3,3)_{-.04} \rightsquigarrow (4,3)_{+1}$
  $(1,1)_{-.04} \rightsquigarrow (1,2)_{-.04} \rightsquigarrow (1,3)_{-.04} \rightsquigarrow (2,3)_{-.04} \rightsquigarrow (3,3)_{-.04} \rightsquigarrow (3,2)_{-.04} \rightsquigarrow (3,3)_{-.04} \rightsquigarrow (4,3)_{+1}$

  $(1,1)_{-.04} \rightsquigarrow (2,1)_{-.04} \rightsquigarrow (3,1)_{-.04} \rightsquigarrow (3,2)_{-.04} \rightsquigarrow (4,2)_{-1}$
- The policy is the same, but the environment is non-deterministic
- The goal is to learn the expected utility $U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t)\right]$

Content

Introduction

Passive learning
- Model-based learning: Adaptive dynamic programming
- Model-free learning: Direct utility estimation
- Model-free learning: Temporal-Difference learning

Active learning
- Q-learning
- Deep Q-learning

Conclusions

Model-based learning: Adaptive dynamic programming (ADP)

1. Learn the transition model
   Estimate $P(s'|s, \pi(s))$ and $R(s)$ from trials.

   Use a table of probabilities: compute how often the result of an action occurs and estimate the transition probability.

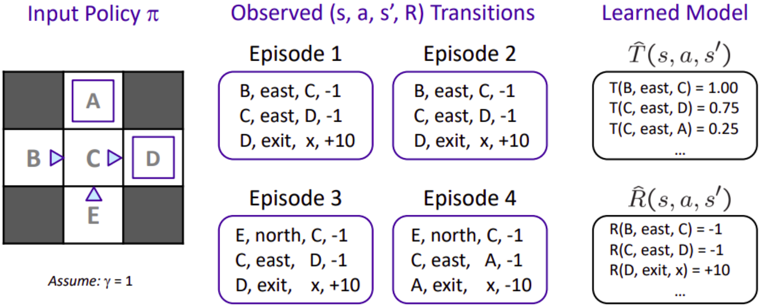
   Example: $(1,1)_{-.04} \rightsquigarrow (1,2)_{-.04} \rightsquigarrow (1,3)_{-.04} \rightsquigarrow (1,2)_{-.04} \rightsquigarrow (1,3)_{-.04} \rightsquigarrow (2,3)_{-.04} \rightsquigarrow (3,3)_{-.04} \rightsquigarrow (4,3)_{+1}$
   $(1,1)_{-.04} \rightsquigarrow (1,2)_{-.04} \rightsquigarrow (1,3)_{-.04} \rightsquigarrow (2,3)_{-.04} \rightsquigarrow (3,3)_{-.04} \rightsquigarrow (3,2)_{-.04} \rightsquigarrow (3,3)_{-.04} \rightsquigarrow (4,3)_{+1}$

   $(1,1)_{-.04} \rightsquigarrow (2,1)_{-.04} \rightsquigarrow (3,1)_{-.04} \rightsquigarrow (3,2)_{-.04} \rightsquigarrow (4,2)_{-1}$
   The action *Right* is executed 3 times in state (1,3) and in 2 cases the result is state (2,3) $\implies P((2,3)|(1,3), Right) = 2/3$

2. Solve MDP

Adaptive dynamic programming (ADP)

1. Learn the empiric model
   Example:



Adaptive dynamic programming

2. Use dynamic programming to solve the MDP.

   Enter the learned probabilities and rewards into the Bellman equations (fixed policy).

   $$U^{\pi}(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi(s)) U^{\pi}(s')$$

   Solve the system of linear equations with the unknowns $U^{\pi}(s)$.

ADP is inefficient if the state space is large

- system of linear equations of order n
- backgammon: 10^{20} equations with 10^{20} unknowns

Content

Introduction

Passive learning
- Model-based learning: Adaptive dynamic programming
- Model-free learning: Direct utility estimation
- Model-free learning: Temporal-Difference learning

Active learning
- Q-learning
- Deep Q-learning

Conclusions

Model-free learning: Direct utility estimation

- The utility of a state is the total expected reward from that state forward (**reward-to-go**)

  Example: $(1,1)_{-.04} \rightsquigarrow (1,2)_{-.04} \rightsquigarrow (1,3)_{-.04} \rightsquigarrow (1,2)_{-.04} \rightsquigarrow (1,3)_{-.04} \rightsquigarrow (2,3)_{-.04} \rightsquigarrow (3,3)_{-.04} \rightsquigarrow (4,3)_{+1}$
  $(1,1)_{-.04} \rightsquigarrow (1,2)_{-.04} \rightsquigarrow (1,3)_{-.04} \rightsquigarrow (2,3)_{-.04} \rightsquigarrow (3,3)_{-.04} \rightsquigarrow (3,2)_{-.04} \rightsquigarrow (3,3)_{-.04} \rightsquigarrow (4,3)_{+1}$
  $(1,1)_{-.04} \rightsquigarrow (2,1)_{-.04} \rightsquigarrow (3,1)_{-.04} \rightsquigarrow (3,2)_{-.04} \rightsquigarrow (4,2)_{-1}$
  The first attempt produces:
  - in state (1,1), total reward 0.72 (1 - .04 x 7)
  - in state (1,2), two total rewards 0.76 and 0.84
  - in state (1,3) two total rewards 0.80 and 0.88

- Estimated utility: **mean** of sampled values
  - U(1,1) = 0.72, U(1,2) = 0.80, U(1,3) = 0.84 etc.

Direct utility estimation

- Assume utilities are independent (false).
  It doesn't take into account that the utility of a state depends on the utilities of next states (the constraints given by the Bellman equations)

$$U^{\pi}(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi(s)) U^{\pi}(s')$$

  - search in a much larger space
  - convergence is very slow

- Have all episodes before

Content

Introduction

Passive learning
- Model-based learning: Adaptive dynamic programming
- Model-free learning: Direct utility estimation
- Model-free learning: Temporal-Difference learning

Active learning
- Q-learning
- Deep Q-learning

Conclusions

Temporal-Difference learning

- Combines the advantages of *Adaptive Dynamic Programming* and *Direct utility estimation*
  - approximately satisfy the Bellman equations
  - update only the directly affected states

- Goal: estimate utilities $U^\pi(s)$, given the episodes generated using policy π. Actions are decided by policy π.

- The utilities are adjusted after each observed transition.

  Example:
  - After 1st trial: the estimates $U^\pi(1,3) = 0.84$, $U^\pi(2,3) = 0.92$.
  - 2nd trial: the transition $(1,3) \rightarrow (2,3)$.
    The constraint given by the Bellman equation requires the update of $U^\pi(1,3)$.

Temporal-Difference learning

- The temporal difference equation uses the difference of utilities between successive states:

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

  α learning rate
  The update involves only the successor s', while the equilibrium conditions (Bellman eq.) involve all possible next states.

- The method applies a series of corrections to converge

- Obs: the method doesn't need a transitions model to perform the updates

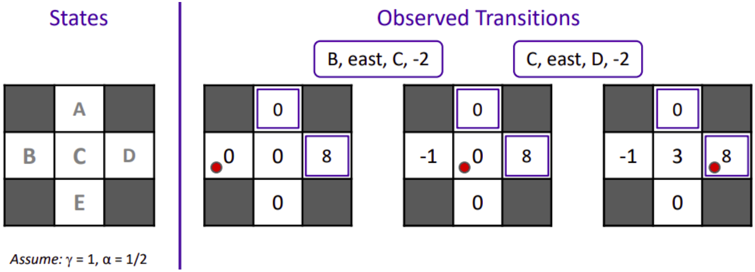
Temporal difference learning: pseudocode

function PASSIVE-TD-AGENT(*percept*) **returns** an action
  inputs: *percept*, a percept indicating the current state s' and reward signal r'
  persistent: π, a fixed policy
    U, a table of utilities, initially empty
    N_s, a table of frequencies for states, initially zero
    s, a, r, the previous state, action, and reward, initially null

  if s' is new **then** $U[s'] \leftarrow r'$
  if s is not null **then**
    increment $N_s[s]$
    $U[s] \leftarrow U[s] + \alpha(N_s[s])(r + \gamma U[s'] - U[s])$
  if s'.TERMINAL? **then** $s, a, r \leftarrow$ null **else** $s, a, r \leftarrow s', \pi[s'], r'$
  return a

Demo: `https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_td.html`

Temporal difference learning: example

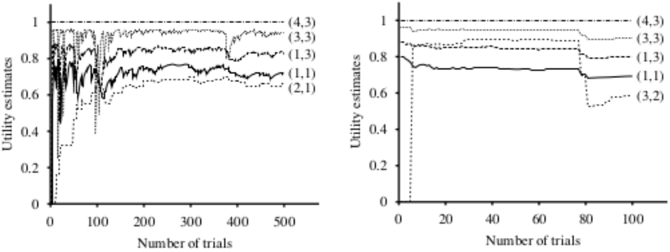


Temporal-Difference learning

- The learning rate α determines the speed of convergence to the true utility
- The mean value of $U^{\pi}(s)$ will converge to the correct value
  - many trials, sparse transitions occur rarely
  - if α is a function which decreases as the no. of visits to a state increases, then $U^{\pi}(s)$ converges to the correct value
    - $\alpha(n) = 1/n$ or $\alpha(n) = 1/(1+n) \in (0, 1]$

Temporal differences learning vs. Adaptive dynamic programming

- TD needs no model, ADP is model-based
- TD use only the observed successor for updating and not all successors
- TD converges more slowly but performs simpler calculations



- TD can be seen as an approximation of ADP

Content

Introduction

Passive learning
- Model-based learning: Adaptive dynamic programming
- Model-free learning: Direct utility estimation
- Model-free learning: Temporal-Difference learning

Active learning
- Q-learning
- Deep Q-learning

Conclusions

Passive learning vs. Active learning

- Passive agent has a fixed policy vs.

  the active agent must decide actions

- Passive agent learns (transition probabilities and) utilities and chooses optimal actions

  vs.

  The active agent updates its policy as it learns
  - the goal is to learn the optimal policy
  - but, the utility function is known approximately

Exploitation vs. exploration

Exploitation-exploration dilemma of the agent

- to maximize his utility, based on the current knowledge, or
- to improve his knowledge

A compromise between

- exploitation
  - the agent stops learning and executes the actions given by the policy
- exploration
  - the agent learns by trying new actions

is necessary.

Exploitation-exploration dilemma: solutions

ϵ-greedy method

- Let $\epsilon \in [0, 1]$
- The next action to be selected will be:
  - a random action, with probability ϵ
  - the optimal action, with probability $1 - \epsilon$

- Implementation
  - initially, $\epsilon = 1$ (exploration)
  - when a learning episode ends, ϵ decreases (for ex. with 0.05) - the exploitation rate progressively increases
  - ϵ never falls bellow a threshold, e.g. 0.1
    - the agent always has a chance to explore, to avoid local optima

Content

Introduction

Passive learning
- Model-based learning: Adaptive dynamic programming
- Model-free learning: Direct utility estimation
- Model-free learning: Temporal-Difference learning

Active learning
- Q-learning
- Deep Q-learning

Conclusions

Algorithm Q-Learning (Watkins, 1989)

- The Q-Learning algorithm learns an action-value function $Q(s, a)$ (Q *quality*).
  $Q(s, a)$ the value of using action a in state s.
  The relationship between utilities and Q values: $U(s) = max_a Q(s, a)$

- The true equations at equilibrium when the Q values are correct

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

  These can be used in an iterative process that computes the exact Q values.

Q-Learning algorithm

- A TD agent which learns a Q function does not need a probabilistic model $P(s'|s,a)$ (*model-free learning*).
- For each sample (s,a,s',r), update the Q value.
  The update equation for *TD Q-Learning*:

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

  (by executing action a in state s we reach the state s')

  The learning coefficient α determines the speed of updating the estimates; usually, $\alpha \in (0,1)$

Q-Learning algorithm: pseudocode

function Q-LEARNING-AGENT(*percept*) **returns** an action
  inputs: *percept*, a percept indicating the current state s' and reward signal r'
  persistent: Q, a table of action values indexed by state and action, initially zero
    N_{sa}, a table of frequencies for state–action pairs, initially zero
    s, a, r, the previous state, action, and reward, initially null

  if TERMINAL?(s) **then** $Q[s, None] \leftarrow r'$
  if s is not null **then**
    increment $N_{sa}[s, a]$
    $Q[s, a] \leftarrow Q[s, a] + \alpha(N_{sa}[s, a])(r + \gamma \max_{a'} Q[s', a'] - Q[s, a])$
  $s, a, r \leftarrow s', \mathrm{argmax}_{a'} f(Q[s', a'], N_{sa}[s', a']), r'$
  return a

f exploration function

- Q-learning converges to an optimal policy
- Q-Learning is slower than ADP

Example 1

Pacman is in an unknown MDP where there are three states [A, B, C] and two actions [Stop, Go]. We are given the following samples generated from taking actions in the unknown MDP. For the following problems, assume $\gamma = 1$ and $\alpha = 0.5$.

(a) We run Q-learning on the following samples:

s	**a**	**s'**	**r**
A	Go	B	2
C	Stop	A	0
B	Stop	A	-2
B	Go	C	-6
C	Go	A	2
A	Go	A	-2

What are the estimates for the following Q-values as obtained by Q-learning? All Q-values are initialized to 0.

$Q(C, Stop) = ?, Q(C, Go) = ?$

Example 2

https://huggingface.co/learn/deep-rl-course/unit2/q-learning-example

SARSA

- The update equation:

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

  (s', a') the pair (next state, next action)

  SARSA uses the TD approach: update the Q table after each step until the solution converges / max. no. of iterations.

- Example: Windy Gridworld
  http://www.incompleteideas.net/book/ebook/node64.html

Content

Introduction

Passive learning
- Model-based learning: Adaptive dynamic programming
- Model-free learning: Direct utility estimation
- Model-free learning: Temporal-Difference learning

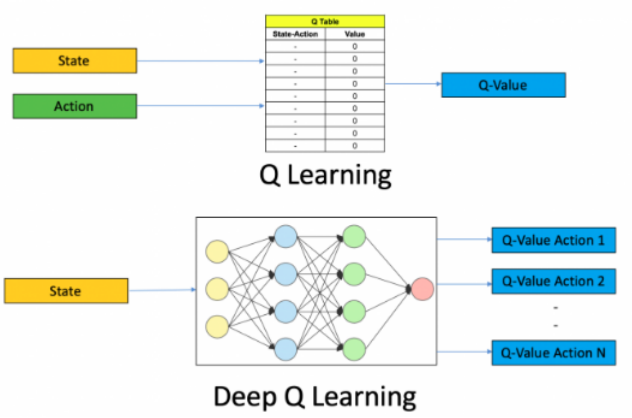
Active learning
- Q-learning
- Deep Q-learning

Conclusions

Deep Reinforcement Learning

It uses a (deep) neural network to approximate the Q values

- input: a state
  output: an estimate of Q, for each possible action



Deep Reinforcement Learning

- Consider the update equation of the Q values:

$$\begin{aligned} Q(s,a) &= Q(s,a) + \alpha[r + \gamma max_{a'} Q(s',a') - Q(s,a)] \\ &= (1-\alpha)Q(s,a) + \alpha[r + \gamma max_{a'} Q(s',a')] \end{aligned}$$

- The *loss function*: the mean squared error of the predicted Q value and the target value Q^* (not known).

  Minimize $loss(s,a,s') = (Q(s,a) - target(s'))^2$, where
  the target value: $target(s') = r + \gamma max_{a'} Q(s',a')$

- Use *Gradient descent* to optimize the loss function.

  Description: `https://deeplearningmath.org/deep-reinforcement-learning.html`
  Demo: https://cs.stanford.edu/people/karpathy/reinforcejs/

Deep Q-learning

Problems:

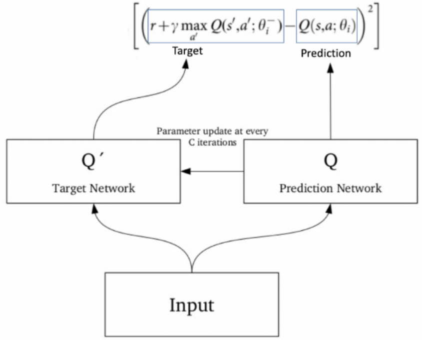
- the samples are correlated $\rightarrow$ the network cannot generalize
- $target(s')$ is an estimate $\rightarrow$ slow convergence/unstable alg.

Solutions:

- ϵ-greedy policy
- *experience replay*: store the experiences (s, a, r, s') and use them for training (*mini-batch*)

Double Deep Q-network

- the target value changes at each iteration; solution: a separate network to estimate the target value



- every C iterations, the parameters from the prediction network are copied to the target network

Function approximation

$$\hat{U}_\theta(s) = \theta_1 f_1(s) + \theta_2 f_2(s) + \ldots \theta_n f_n(s)$$

$f_1, \ldots f_n$ features

RL learn values for the parameters $\theta = \theta_1, \ldots \theta_n$ s.t the evaluation function $\hat{U}_\theta$ approximates the true utility function.

- updates the parameters after each trial
- use an error function and compute the gradients

$$E_j(s) = (\hat{U}_\theta(s) - u_j(s))^2/2$$

$u_j(s)$ the observed total reward from state s in trial j

$$\theta_i \leftarrow \theta_i - \alpha \frac{\delta E_j(s)}{\delta \theta_i} = \theta_i + \alpha(u_j(s) - \hat{U}_\theta(s)) \frac{\delta \hat{U}_\theta(s)}{\delta \theta_i}$$

- good generalization (visited states $\rightarrow$ unvisited states)

Content

Introduction

Passive learning
- Model-based learning: Adaptive dynamic programming
- Model-free learning: Direct utility estimation
- Model-free learning: Temporal-Difference learning

Active learning
- Q-learning
- Deep Q-learning

Conclusions

Conclusions

- Reinforcement learning is necessary for agents that evolve in unfamiliar environments
- Passive learning evaluates a given policy
- Active learning learns an optimal policy

Bibliography

- *Artificial Intelligence: A modern Approach*. Ch. 21. Reinforcement Learning
- Sutton&Barto. *Reinforcement Learning*. An introduction `http://incompleteideas.net/book/RLbook2020.pdf`