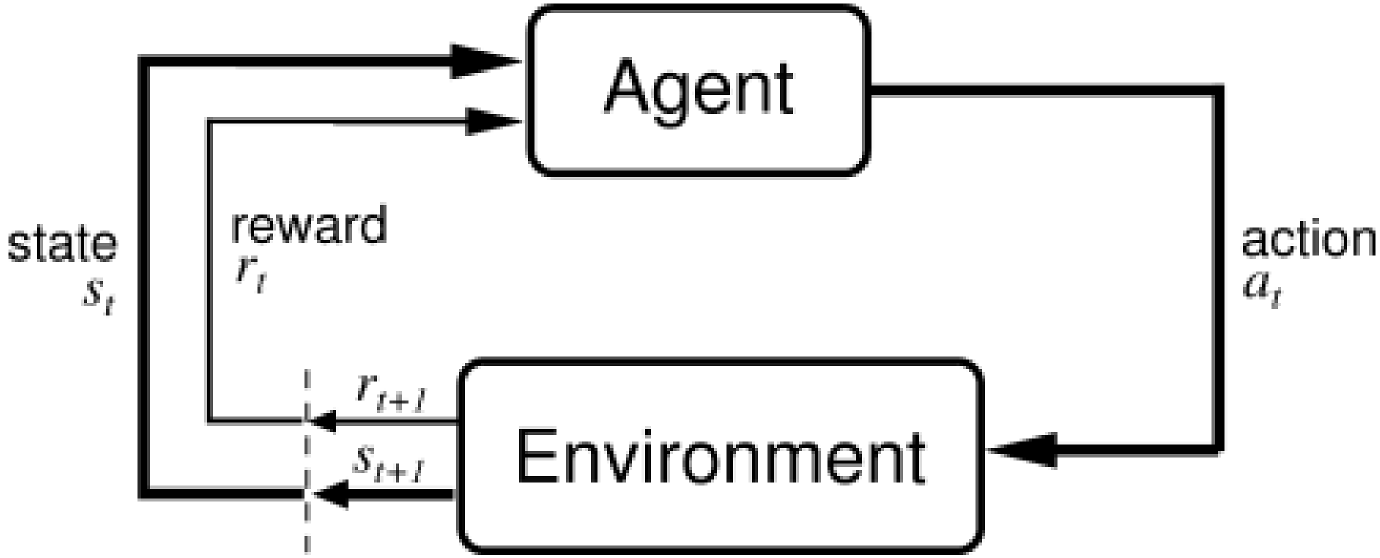
Chapter 7: Reinforcement Learning

Hankui Zhuo

March 29, 2019

Reinforcement Learning

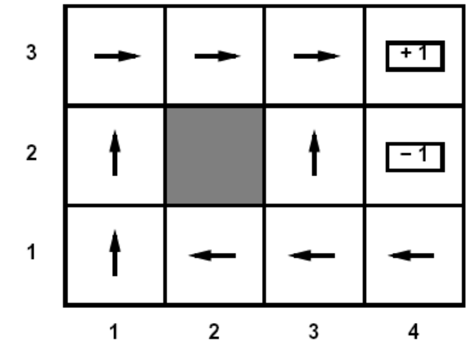
- Basic idea:
  - Receive feedback in the form of rewards
  - Agent's utility is defined by the reward function
  - Must (learn to) act so as to maximize expected rewards



Reinforcement Learning

- Reinforcement learning:
  - Still assume an MDP:
    - A set of states $s \in S$
    - A set of actions (per state) A
    - A model T(s,a,s')
    - A reward function R(s,a,s')
    - A discount factor γ (could be 1)
  - Still looking for a policy $\pi(s)$
  - New twist: don't know T or R
    - i.e. don't know which states are good or what the actions do
    - Must actually try actions and states out to learn

Passive Learning



- Simplified task
  - You don't know the transitions T(s,a,s')
  - You don't know the rewards R(s,a,s')
  - You are given a policy $\pi(s)$
  - Goal: learn the state values

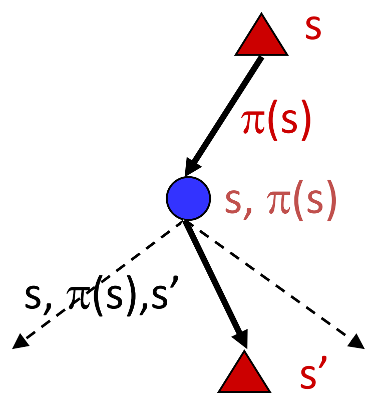
- In this case:
  - No choice about what actions to take
  - Just execute the policy and learn from experience
  - This is NOT offline planning! You actually take actions in the world and see what happens…

Model-Based Learning

- Idea:
  经验
  - Learn the model empirically through experience
  - Solve for values as if the learned model were correct

- Simple empirical model learning
  - Count outcomes for each s,a
  - Normalize to give estimate of **T(s,a,s')**
  - Discover **R(s,a,s')** when we experience (s,a,s')

- Solving the MDP with the learned model
  - Iterative policy evaluation, for example

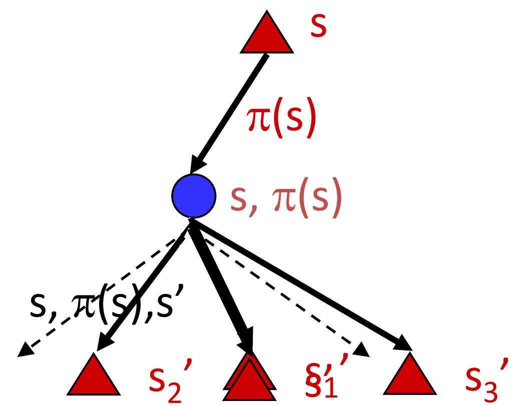


$$V_{i+1}^{\pi}(s) \leftarrow \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma V_i^{\pi}(s')]$$

Sample-Based Policy Evaluation?

$$V_{i+1}^{\pi}(s) \leftarrow \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V_i^{\pi}(s')]$$

- Approximate the expectation with samples (drawn from T!)



$$sample_1 = R(s, \pi(s), s_1') + \gamma V_i^{\pi}(s_1')$$

$$sample_2 = R(s, \pi(s), s_2') + \gamma V_i^{\pi}(s_2')$$

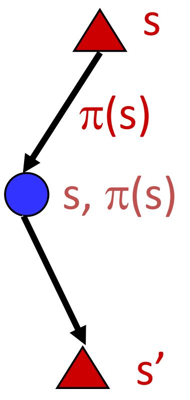
$$\ldots$$

$$sample_k = R(s, \pi(s), s_k') + \gamma V_i^{\pi}(s_k')$$

$$V_{i+1}^{\pi}(s) \leftarrow \frac{1}{k} \sum_i sample_i$$

Temporal-Difference Learning

- Big idea: learn from every experience!
  - Update V(s) each time we experience (s,a,s',r)
  - Likely s' will contribute updates more often



- Temporal difference learning
  - Policy still fixed!
  - Move values toward value of whatever successor occurs: running average!

Sample of V(s): $$sample = R(s, \pi(s), s') + \gamma V^{\pi}(s')$$

Update to V(s): $$V^{\pi}(s) \leftarrow (1 - \alpha)V^{\pi}(s) + (\alpha) sample$$

Exponential Moving Average

- Exponential moving average
  - Makes recent samples more important

$$\bar{x}_n = \frac{x_n + (1-\alpha) \cdot x_{n-1} + (1-\alpha)^2 \cdot x_{n-2} + \ldots}{1 + (1-\alpha) + (1-\alpha)^2 + \ldots}$$

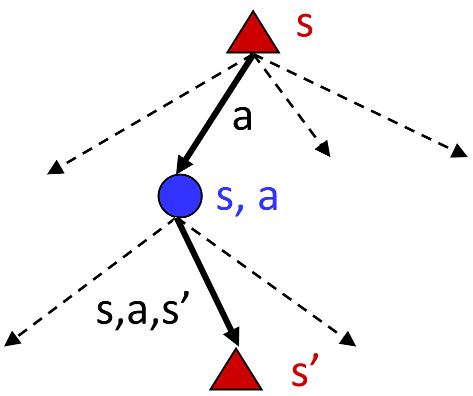
  - Forgets about the past (distant past values were wrong anyway)
  - Easy to compute from the running average

$$\bar{x}_n = (1-\alpha) \cdot \bar{x}_{n-1} + \alpha \cdot x_n$$

- Decreasing learning rate can give converging averages

Problems with TD Value Learning

- TD value leaning is a model-free way to do policy evaluation
- However, if we want to turn values into a (new) policy, we're sunk:

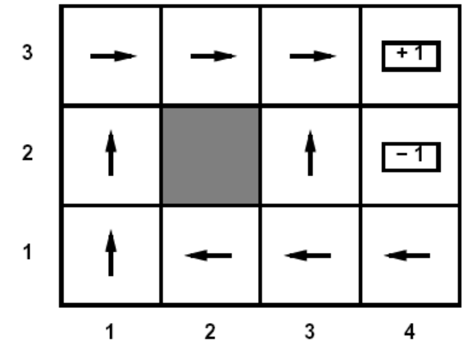


$$\pi(s) = \arg\max_a Q^*(s,a)$$

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

- Idea: learn Q-values directly
- Makes action selection model-free too!

Active Learning



- Full reinforcement learning
  - You don't know the transitions T(s,a,s')
  - You don't know the rewards R(s,a,s')
  - You can choose any actions you like
  - Goal: learn the optimal policy

- In this case:
  - Learner makes choices!
  - Fundamental tradeoff: exploration vs. exploitation
  - This is NOT offline planning! You actually take actions in the world and find out what happens…

Q-Learning

- 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:

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

Q-Learning Properties

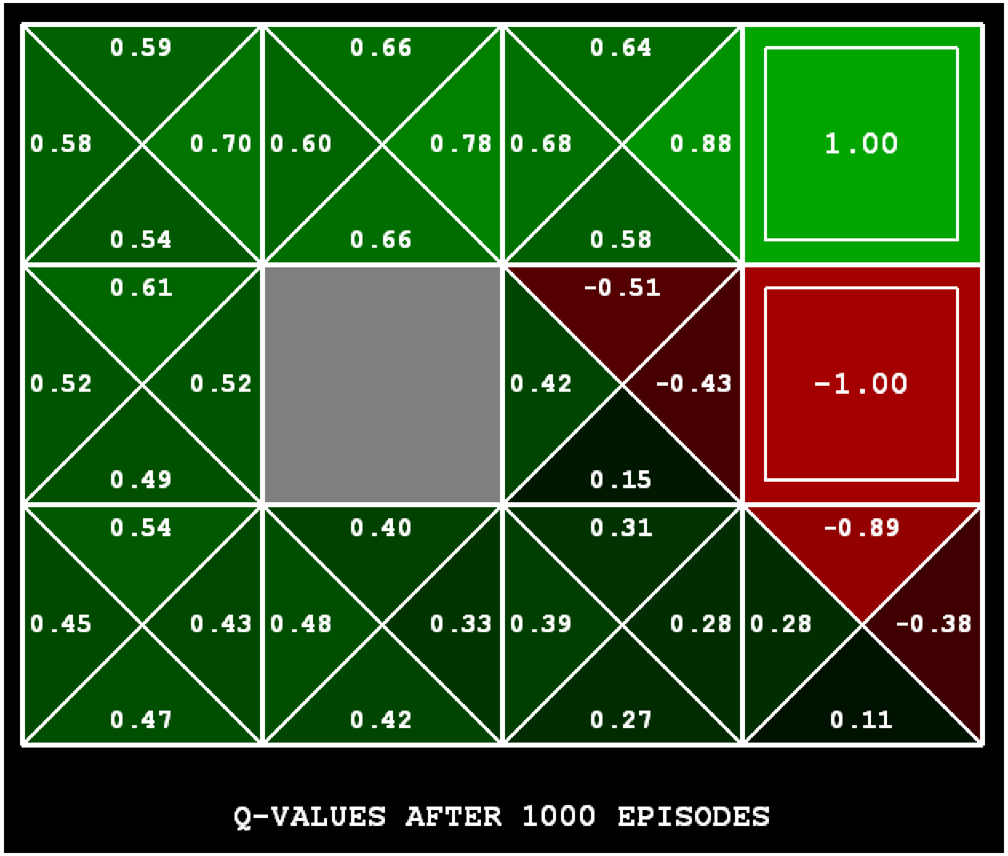
- Amazing result: Q-learning converges to optimal policy
  - If you explore enough
  - If you make the learning rate small enough
  - … but not decrease it too quickly!
  - Basically doesn't matter how you select actions (!)

Exploration / Exploitation

- Several schemes for forcing exploration
  - Simplest: random actions (ε greedy)
    - Every time step, flip a coin
    - With probability ε, act randomly
    - With probability 1-ε, act according to current policy
  - Problems with random actions?
    - You do explore the space, but keep thrashing around once learning is done
    - One solution: lower ε over time

Q-Learning

- Q-learning produces tables of q-values:



The Story So Far: MDPs and RL

Things we know how to do:

- If we know the MDP
  - Compute V^*, Q^*, π^* exactly
  - Evaluate a fixed policy π
- If we don't know the MDP
  - We can estimate the MDP then solve
  - We can estimate V for a fixed policy π
  - We can estimate $Q^*(s,a)$ for the optimal policy while executing an exploration policy

Techniques:

- Model-based DPs
  - Value and policy Iteration
  - Policy evaluation
- Model-based RL
- Model-free RL:
  - Value learning
  - Q-learning

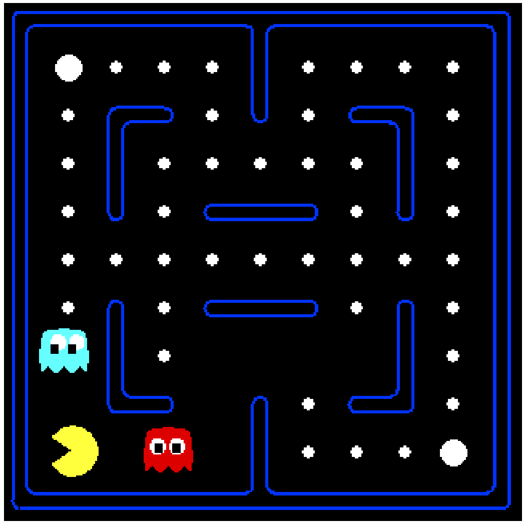
Q-Learning

- In realistic situations, we cannot possibly learn about every single state!
  - Too many states to visit them all in training
  - Too many states to hold the q-tables in memory

- Instead, we want to generalize:
  - Learn about some small number of training states from experience
  - Generalize that experience to new, similar states
  - This is a fundamental idea in machine learning

Feature-Based Representations

- Solution: describe a state using a vector of features
  - Features are functions from states to real numbers (often 0/1) that capture important properties of the state
  - Example features:
    - Distance to closest ghost
    - Distance to closest dot
    - Number of ghosts
    - 1 / (dist to dot)^2
    - Is Pacman in a tunnel? (0/1)
    - …… etc.
  - Can also describe a q-state (s, a) with features (e.g. action moves closer to food)



Linear Feature Functions

- Using a feature representation, we can write a q function (or value function) for any state using a few weights:

$$V(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$$

- Advantage: our experience is summed up in a few powerful numbers
- Disadvantage: states may share features but be very different in value!

Function Approximation

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$$

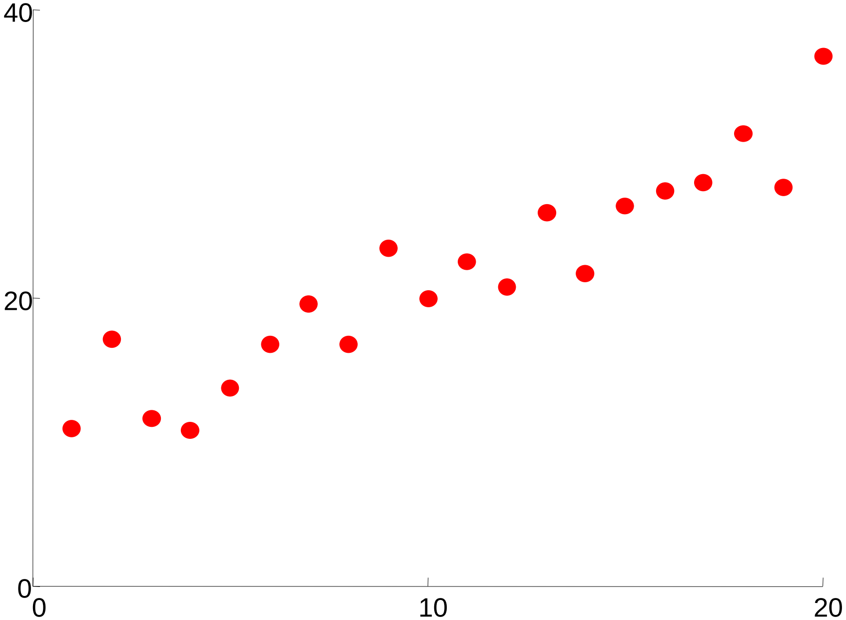
- Q-learning with linear q-functions:

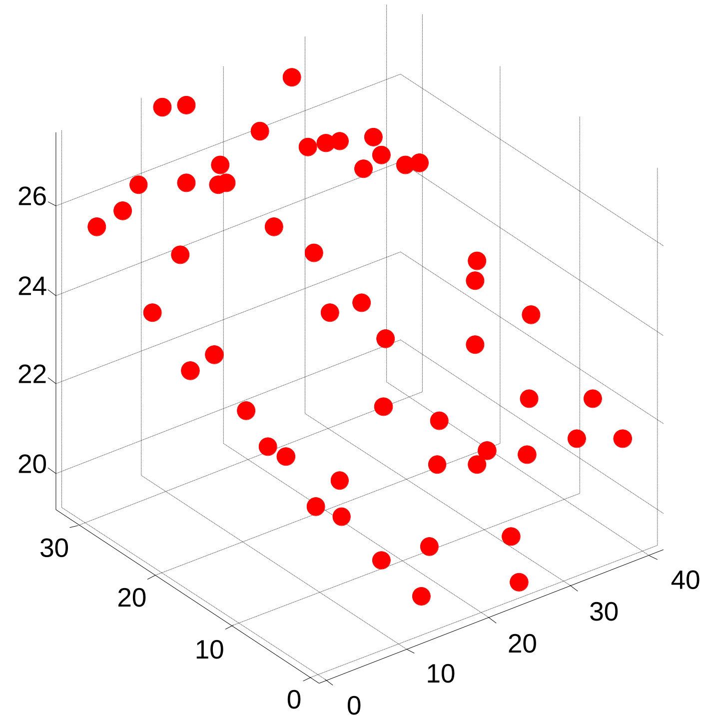
$$Q(s,a) \leftarrow Q(s,a) + \alpha \, [error]$$

$$w_i \leftarrow w_i + \alpha \, [error] \, f_i(s,a)$$

- Intuitive interpretation:
  - Adjust weights of active features
  - E.g. if something unexpectedly bad happens, disprefer all states with that state’s features

Linear regression

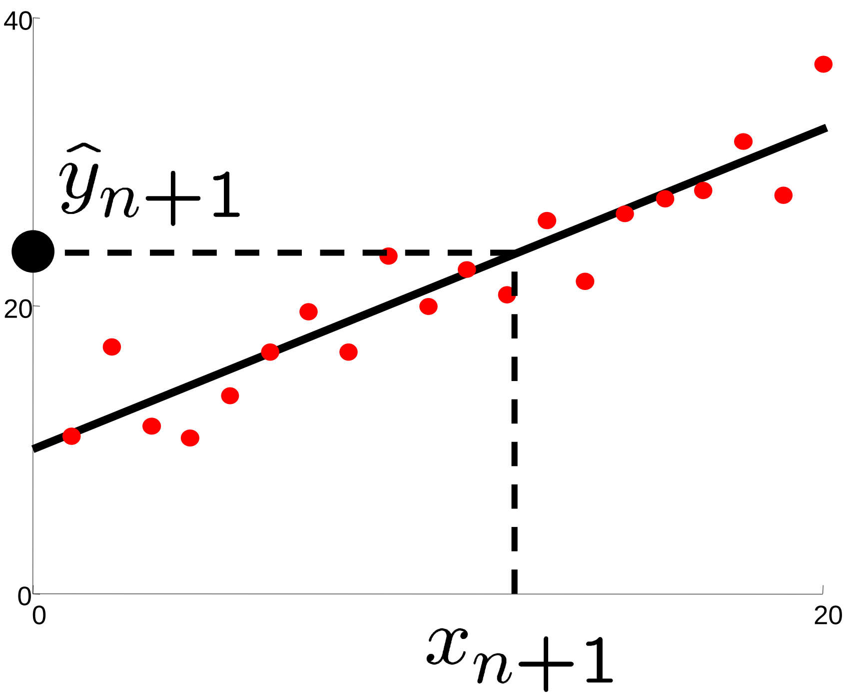


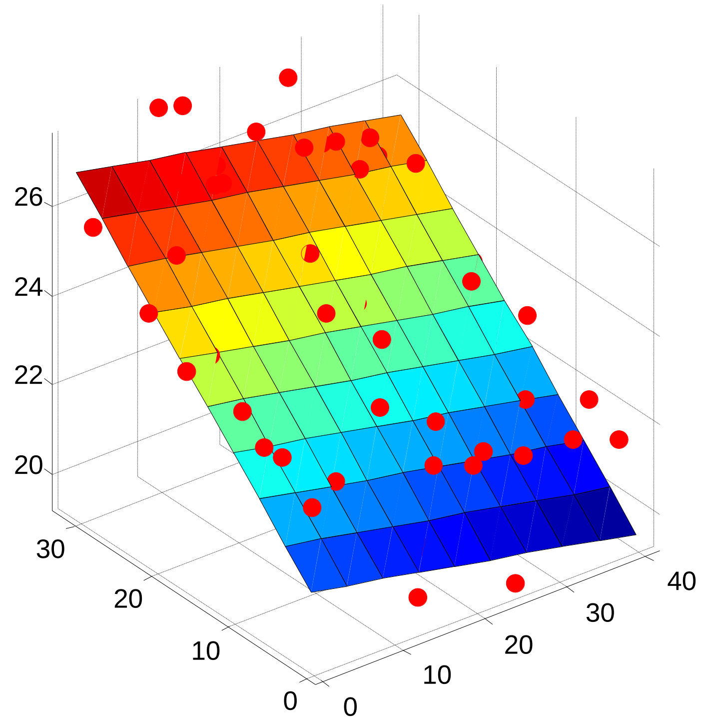


Given examples $(x_i, y_i)_{i=1 \ldots n}$

Predict y_{n+1} given a new point x_{n+1}

Linear regression





Prediction

$$\widehat{y}_i = w_0 + w_1 x_i$$

Prediction

$$\widehat{y}_i = w_0 + w_1 x_{i,1} + w_2 x_{i,2}$$

The End!