# Reinforcement Learning

* Still assume a MDP
  + Set of states S
  + Set of actions A
  + Transimtion model P(s’ | s, a)
  + Reward function R( s, a, s’)
* Still looking for a policy /pi (s)
* New twist:
  + Don’t know P and R
  + We don’t know which states are good or what the actions do
  + Must actually try out actions and states to learn
* Basic idea:
  + Receive feedback in form of rewards
  + Must learn to act so as to maximnize expected rewards
  + All learning is based on framework

Simple gridworld

Episode:

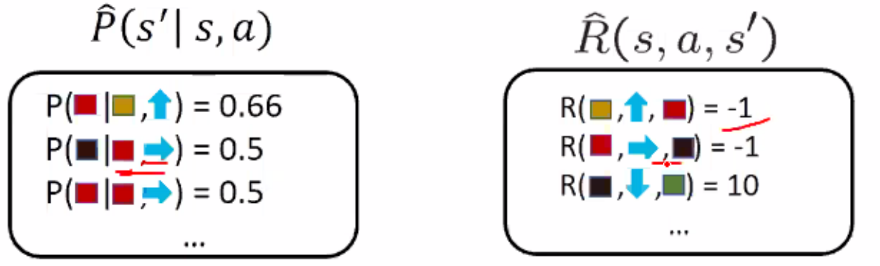
* s0, a0, r1, s1, a1, r2,…st

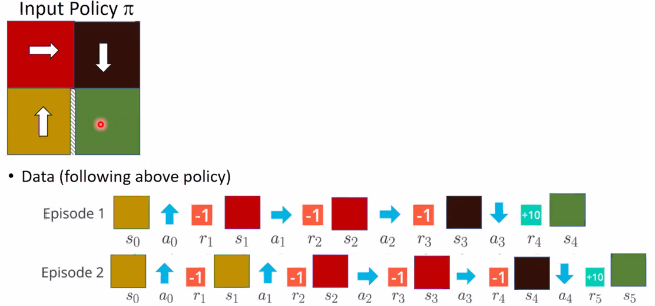
## Model-Based Learning

Monte Carlo

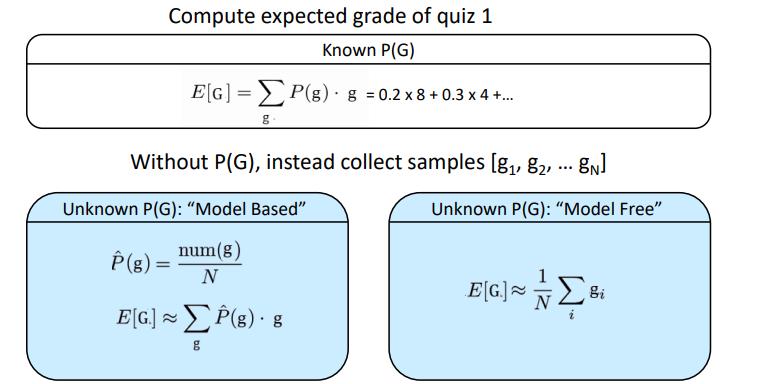
1. Learn empiricical MDP using Monte Carlo simulation data
   * s0, a0, r1, s1, a1, r2,…st
   * Estimate transition and rewards
     1. P(s;, s, a) = Times (s, a, s’) occurs / Times (s, a) occurs
     2. Discover each R(s, a, s’) when we experience (s, a, s’)
   * Estimates converge to true values under certain conditions
2. Solve the learned MDP
   * For example, compute policy using value iteration

Example:





# Model Free Learning



## Passive Reinforcement Learning

Simplified task

* + Input: a fixed policy π(s)
  + Don’t know transitions P(s’ |s, a)
  + Don’t know rewards R(s, a, s’)
  + Goal: Learn values for each state under π

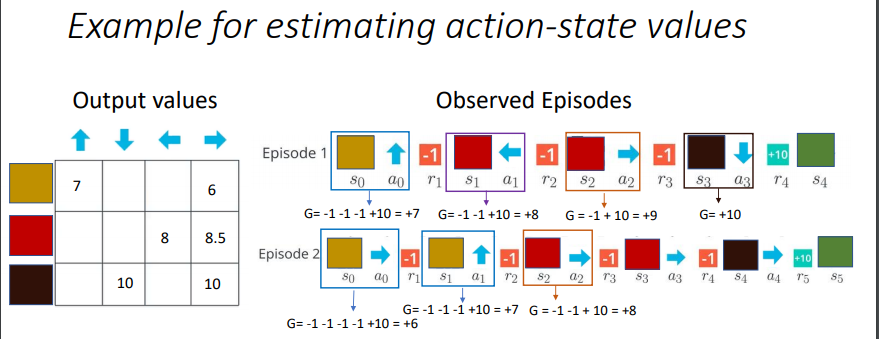
Learner is “along for the ride”

No choice about what actions to take

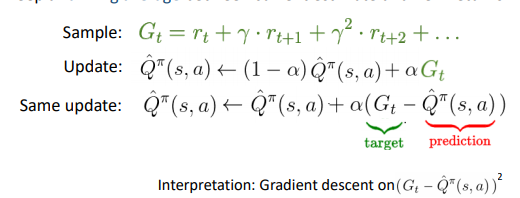
Just execute policy and learn from experience

Direct evaluation (model free monte carlo)

* Average together observed sample values
  + Act according to π and collect data: s0, a0, r1, s1, a1, r2,…st
  + First time you visit a state, write down sum of discounted rewards (G)
  + Average those samples to compute Vpi(s)
  + Same idea for estimating Qpi(s, a), focusing though on q-states



Incremental mean

* Data (following policy pi)
* Qpi(s, a) = average of G, where st = s, at = a
* Alternate formulation:
  + Update Qpionce a Gt has been computed (episode ends)
  + Keep a running average between current estimate and new returns

The good

* Easy to understand
* Doesn’t require P or R
* Eventually computes correct average values, using just sample transitions (no bias)

Bad

* Doesn’t recognize underlying model is an MDP
* Each state must be learned separately. Takes a long time to learn.
* Need to wait until end of an episode to update values
* High variance

## Temporal Difference Learning

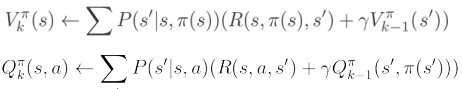
Q/pi(s, a) is expected utility starting in s, taking action a, and then following policy /pi

* 

Monte Carlo Evaluation

* Act according to /pi and collect data: S0, a0, r1, s1, a1, r2,….,sT
* Q/pi(s, a) = average of Gt where st = s, at = a

Recall: policy evaluation?

* 

Big Idea: Learn from every experience

* Update Q/pi each time we experience a transition (s, a, r, s’, a’)
* Keep a running average between current estimate and new experiences
* 
* 

Above is known as SARSA

* s, a, r, s’, a’
* We can use TD-learning to estimate V/pi based on (s, a, r, s’) transitions
* 

# Active Reinforcement Learning

Full reinforcement learning

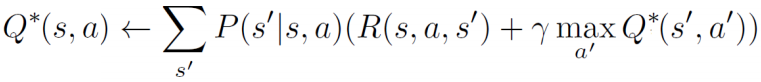
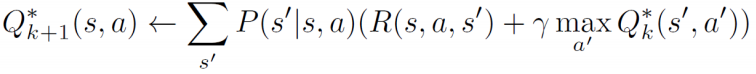
* Don’t know transitions P(s’ | s, a)
* Don’t know rewards R(s, a, s’)
* You choose actions
* Goal: Learn optimal policy / values

Learner makes choices

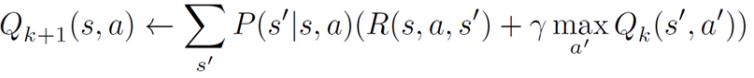
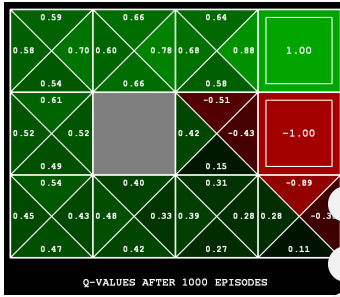
Tradeoff: Exploration vs exploitation

Not offline. Actually take actions in the world

Recap: Q-Values

* Q\*(s, a) = expected utility starting in s, taking action a and (thereafter) acting optimally
* Bellman equation:
  + 
* Q-Value Iteration
  + 
  + 

Tabular Q-Learning

* Q-Learning: Sample-based Q-value iteration
* 

Learn Q-Values as you go

* Receive a sample (s, a, s’, r)
* Consider previous estimate: Q-hat(s, a)
* Consider new estimate:
  + 
* Incorporate new estimate into a running average
  + 