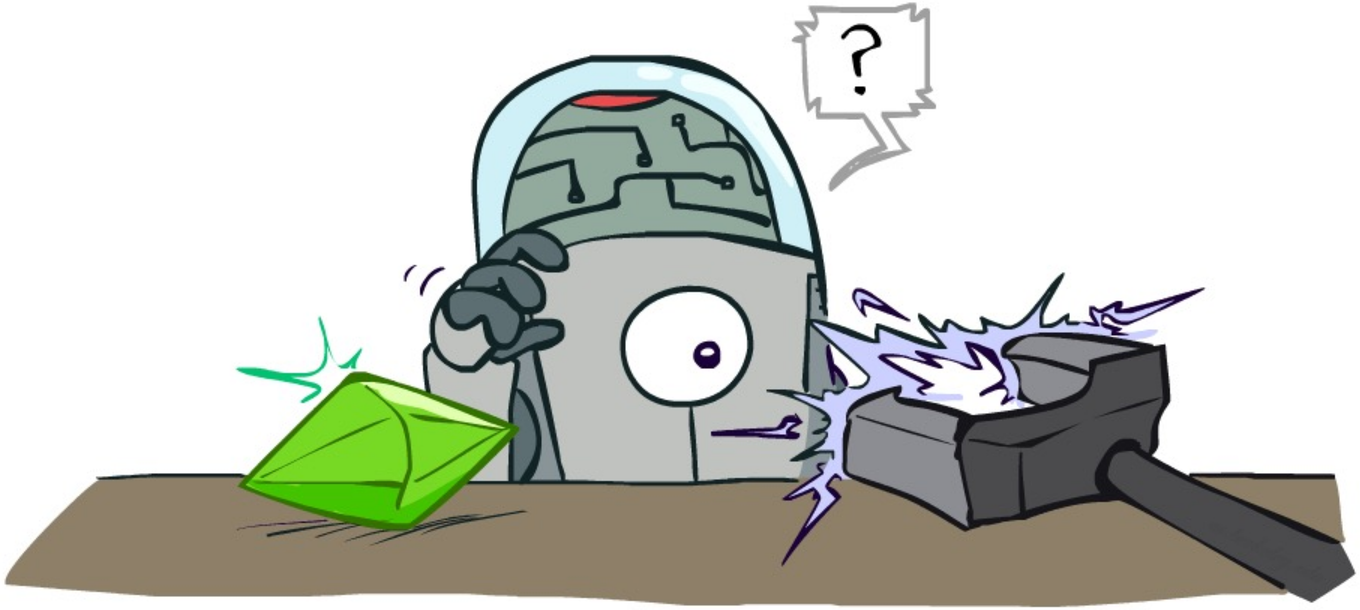


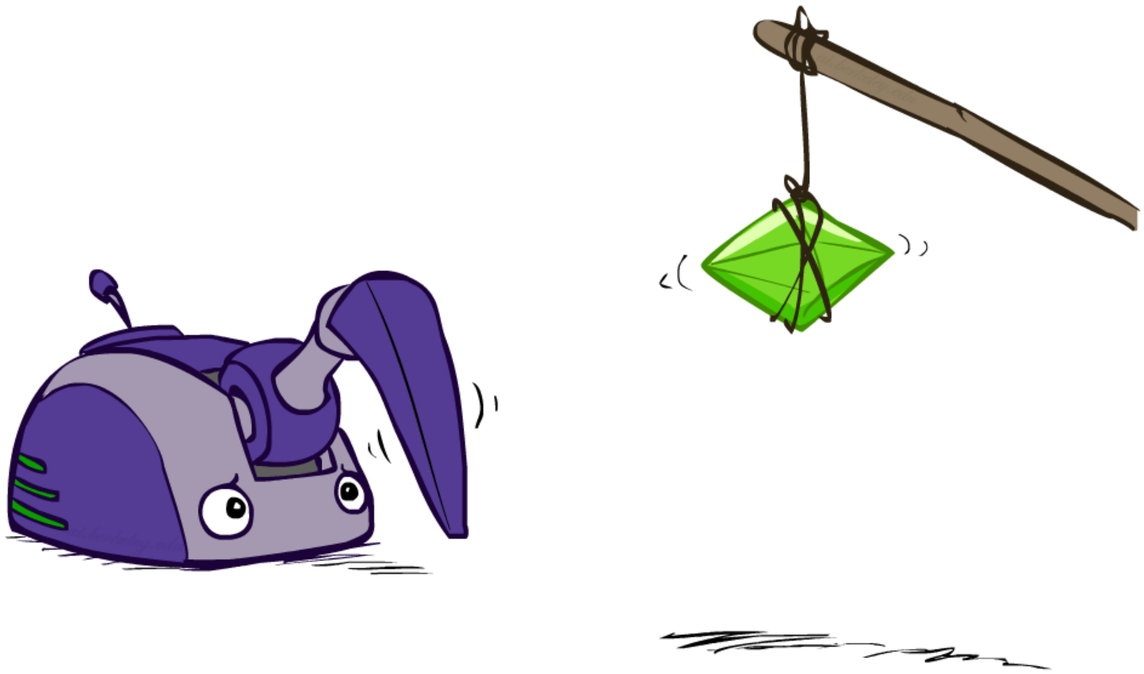
Reinforcement Learning

CE417: Introduction to Artificial Intelligence
Sharif University of Technology
Fall 2023

Soleymani

Slides have been adopted from Klein and Abdeel, CS188, UC Berkeley.

Reinforcement Learning

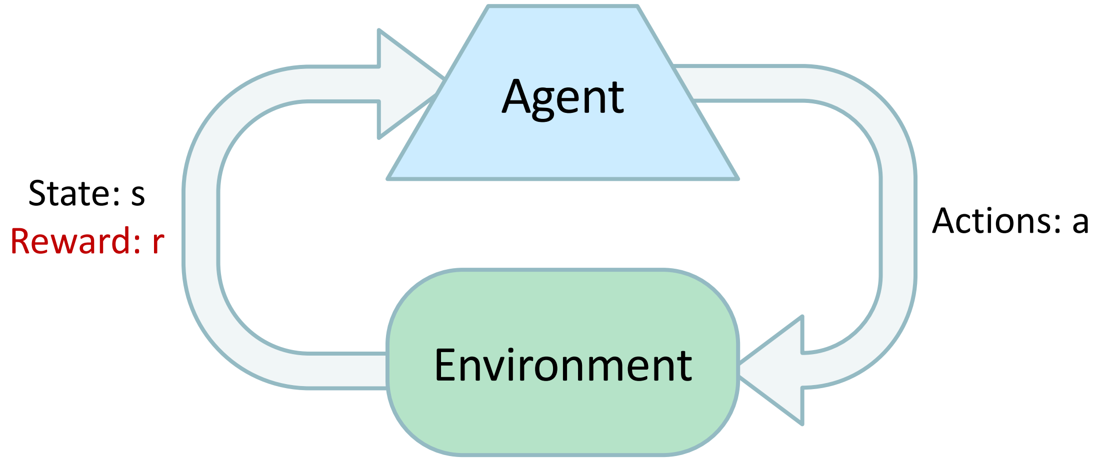


Reinforcement Learning

- Still assume a Markov decision process (MDP):
  - A set of states $s \in S$
  - A set of actions (per state) A(s)
  - A model T(s,a,s')
  - A reward function R(s,a,s')
- Still looking for a policy $\pi(s)$

- New twist: don't know T or R
  - I.e. we don't know which states are good or what the actions do
  - Must actually try actions and states out to learn

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
  - All learning is based on observed samples of outcomes!

Example: Samuel's Checker Player (1956-67)

Example: Learning to Walk

Initial

A Learning Trial

After Learning [1K Trials]

[Kohl and Stone, ICRA 2004]

Example: Learning to Walk

[Kohl and Stone, ICRA 2004]

Initial

[Video: AIBO WALK – initial]

Example: Learning to Walk

[Kohl and Stone, ICRA 2004]

Training

[Video: AIBO WALK – training]

Example: Learning to Walk

[Kohl and Stone, ICRA 2004]

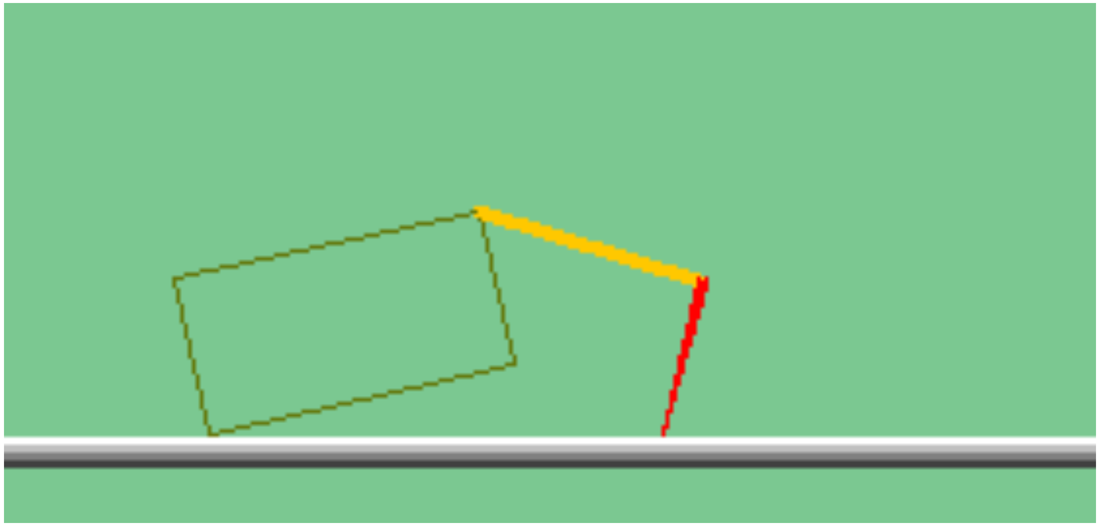
Finished

[Video: AIBO WALK – finished]

Example: Sidewinding

[Andrew Ng]

The Crawler!

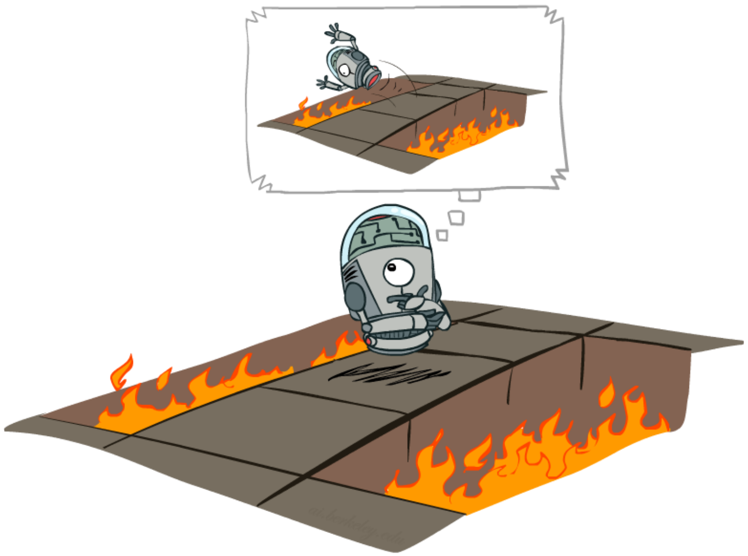


Video of Demo Crawler Bot

Example: Breakout (DeepMind)

[© TwoMinuteLectures]

Offline (MDPs) vs. Online (RL)



Offline Solution

Online Learning

RL vs. MDP

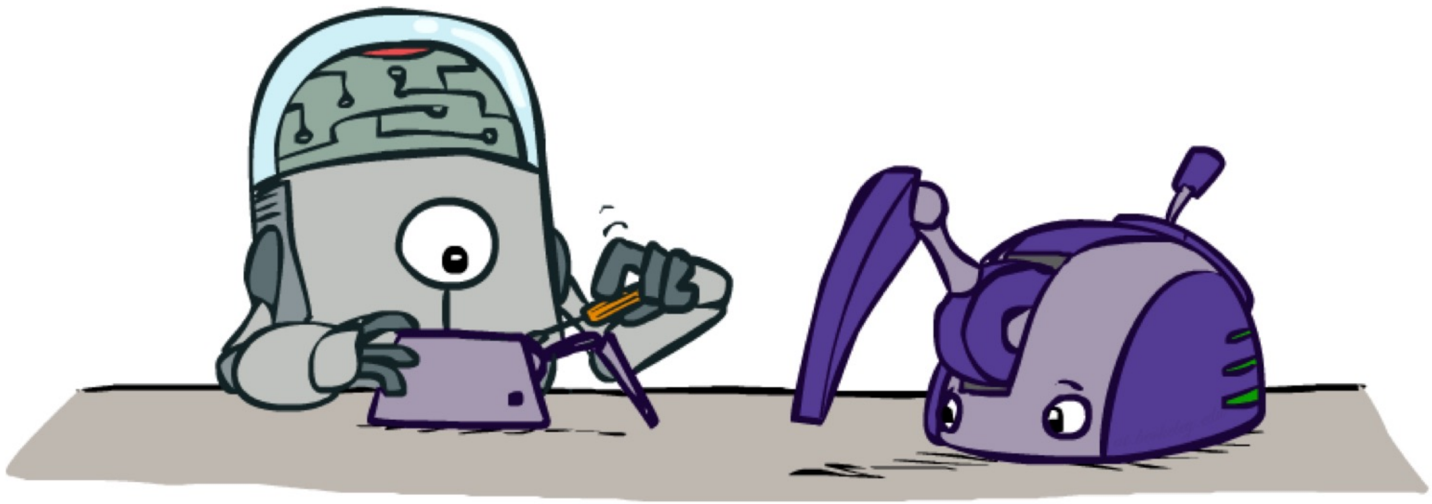
- RL isn't just planning, it is also learning!
  - There is an MDP, but you can't solve it with just computation
  - You need to actually act to figure it out

- Important ideas in reinforcement learning that came up
  - ***Exploration***: you have to ***try unknown actions*** to get information
  - ***Exploitation***: eventually, you have to use what you know
  - ***Regret***: early on, you inevitably "make mistakes" and lose reward
  - ***Sampling***: you may need to repeat many times to get good estimates
  - ***Generalization***: what you learn in one state may apply to others too

Approaches to Reinforcement Learning

- Model-based: Learn the model, solve it, execute the solution
- Learn values from experiences, use to make decisions
  - Direct evaluation
  - Temporal difference learning
  - Q-learning
- Learn policies directly

Model-Based RL



Model-Based Learning

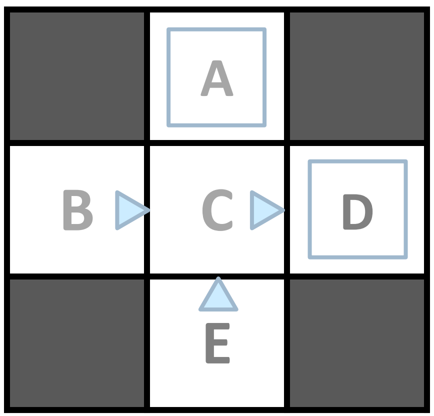
- Model-Based Idea:
  - Learn an approximate model based on experiences
  - Solve for values as if the learned model were correct

- Step 1: Learn empirical MDP model
  - Count outcomes s' for each s, a
  - Normalize to give an estimate of $\hat{T}(s, a, s')$
  - Discover each $\hat{R}(s, a, s')$ when we experience the transition

- Step 2: Solve the learned MDP
  - For example, use value iteration, as before

Example: Model-Based Learning

Input Policy π



Assume: γ = 1

Observed Episodes (Training)

Episode 1

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 2

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 3

E, north, C, -1
C, east, D, -1
D, exit, x, +10

Episode 4

E, north, C, -1
C, east, A, -1
A, exit, x, -10

Learned Model

$\widehat{T}(s, a, s')$

T(B, east, C) = 1.00
T(C, east, D) = 0.75
T(C, east, A) = 0.25
…

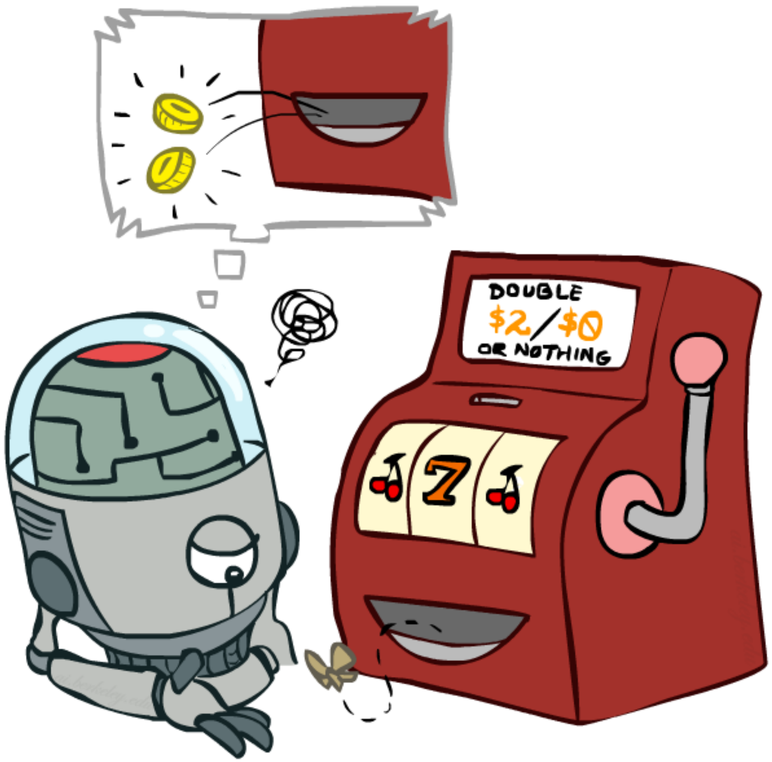
$\widehat{R}(s, a, s')$

R(B, east, C) = -1
R(C, east, D) = -1
R(D, exit, x) = +10
…

Pros and cons

- Pro:
  - Makes efficient use of experiences (low ***sample complexity***)
- Con:
  - May not scale to large state spaces
    - Learns model one state-action pair at a time (but this is fixable)
    - Cannot solve MDP for very large $|S|$ (also somewhat fixable)
  - Much harder when the environment is partially observable

Model-Free Learning



Reinforcement Learning

- We still assume an MDP:
  - A set of states $s \in S$
  - A set of actions (per state) A(s)
  - A model T(s,a,s')
  - A reward function R(s,a,s')
- Still looking for a policy $\pi(s)$

- New twist: don't know T or R, so must try out actions
- Big idea: Compute all averages over T using sample outcomes

Example: Expected Age

Goal: Compute expected age of cs188 students

Known P(A)

$$E[A] = \sum_a P(a) \cdot a \qquad = 0.35 \times 20 + \ldots$$

Without P(A), instead collect samples [a_1, a_2, … a_N]

Unknown P(A): “Model Based”

$$\hat{P}(a) = \frac{\text{num}(a)}{N}$$

$$E[A] \approx \sum_a \hat{P}(a) \cdot a$$

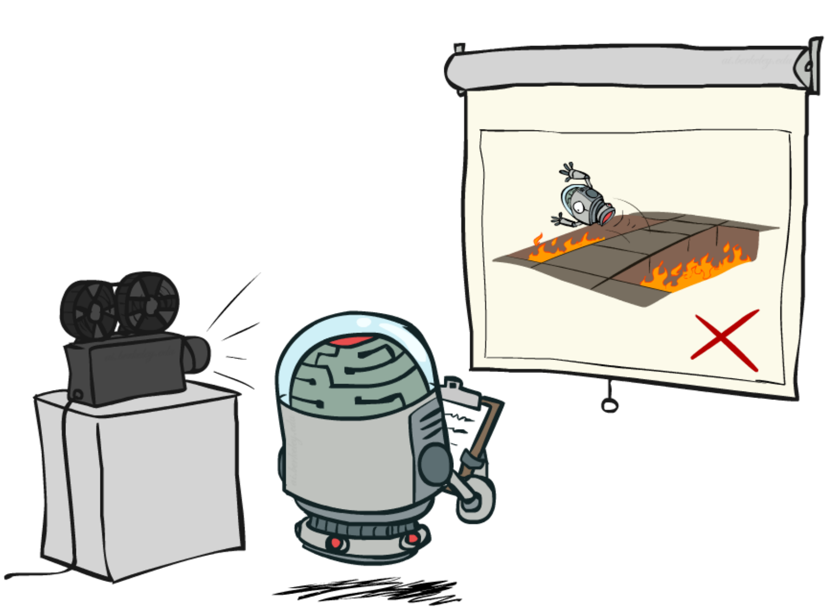
Why does this work? Because eventually you learn the right model.

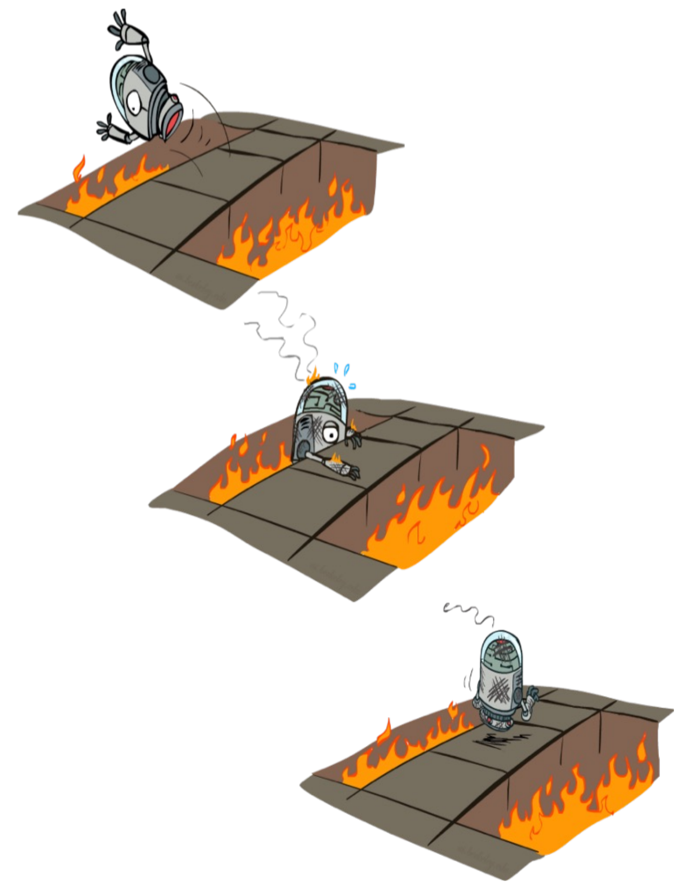
Unknown P(A): “Model Free”

$$E[A] \approx \frac{1}{N} \sum_i a_i$$

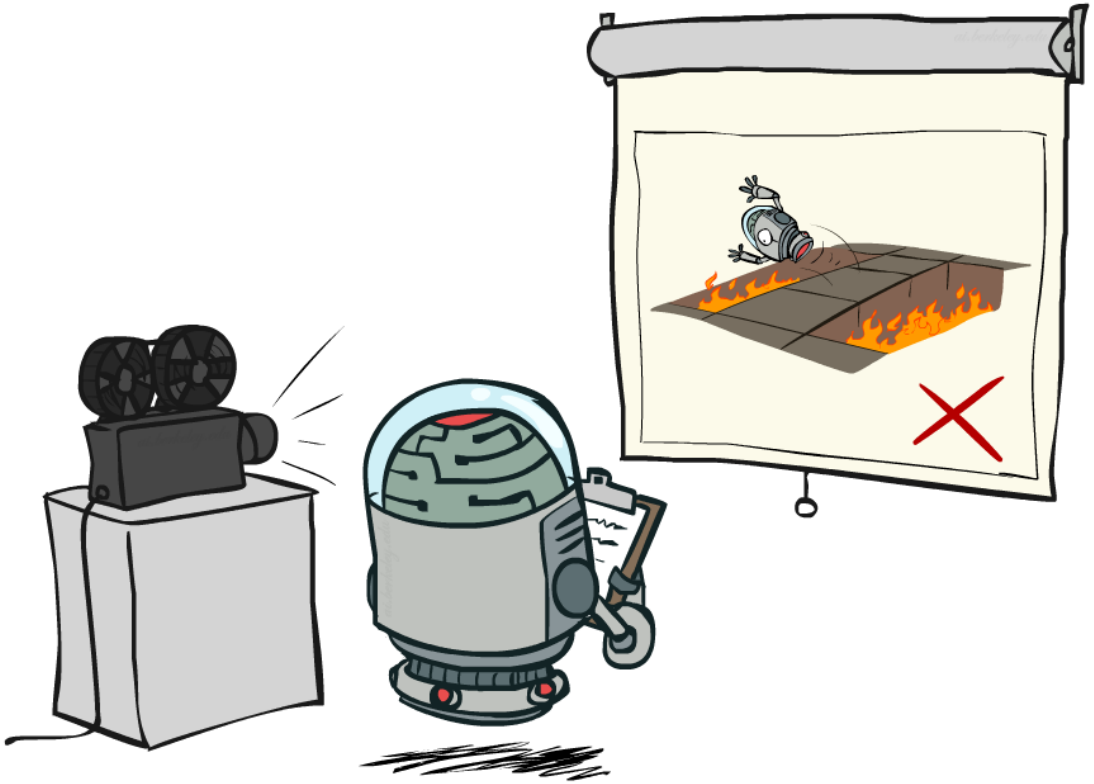
Why does this work? Because samples appear with the right frequencies.

Passive vs. Active RL

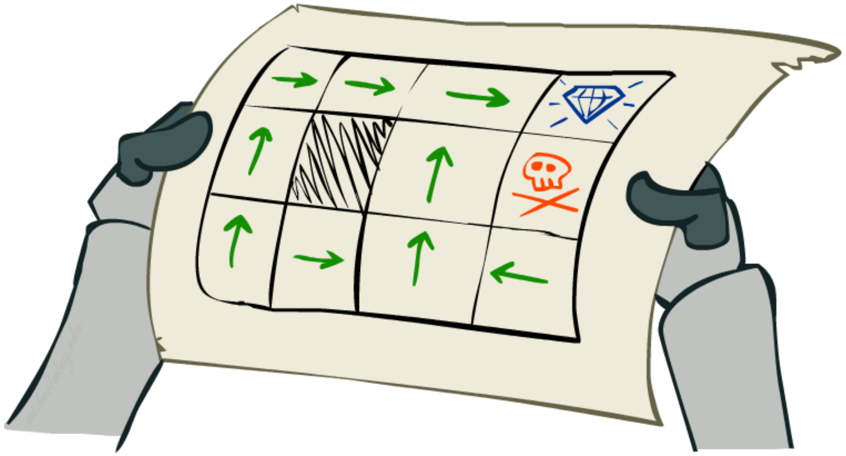




Passive Reinforcement Learning



Passive Reinforcement Learning



- Simplified task: policy evaluation
  - Input: a fixed policy $\pi(s)$
  - You don't know the transitions T(s,a,s')
  - You don't know the rewards R(s,a,s')
  - Goal: learn the state values $V^{\pi}(s)$

- In this case:
  - Learner is "along for the ride"
  - 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.

Direct Evaluation (Monte Carlo)

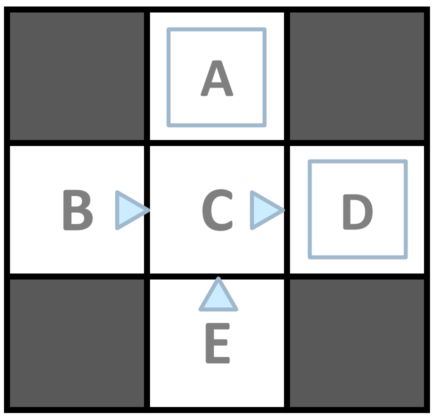
- Goal: Estimate $V^{\pi}(s)$, i.e., expected total discounted reward from s onwards

- Idea: Average together observed sample values
  - Act according to π
  - Every time you visit a state, write down what the sum of discounted rewards turned out to be
  - Average those samples

- This is called direct evaluation by Monte Carlo estimation (or direct utility estimation)

Example: Direct Evaluation

Input Policy π



Assume: γ = 1

Observed Episodes (Training)

Episode 1

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 2

B, east, C, -1
C, east, D, -1
D, exit, x, +10

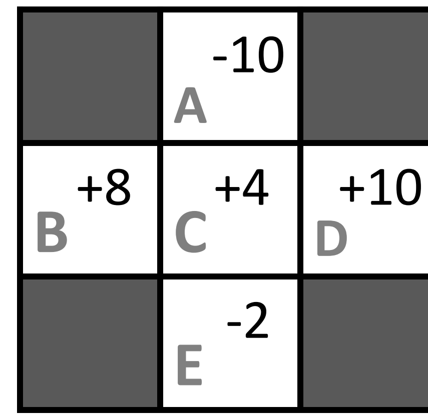
Episode 3

E, north, C, -1
C, east, D, -1
D, exit, x, +10

Episode 4

E, north, C, -1
C, east, A, -1
A, exit, x, -10

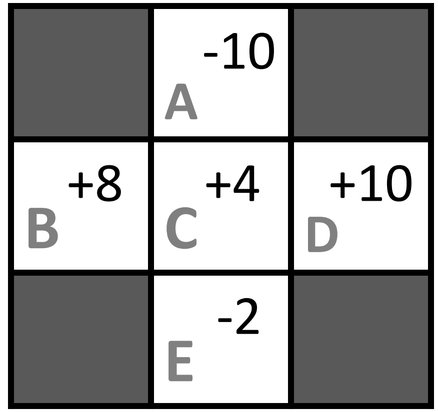
Output Values



Problems with Direct Evaluation

- What's good about direct evaluation?
  - It's easy to understand
  - It doesn't require any knowledge of T, R
  - It eventually computes the correct average values, using just sample transitions
- What's bad about it?
  - It ignores information about state connections
  - Each state must be learned separately
  - So, it takes a long time to learn

Output Values



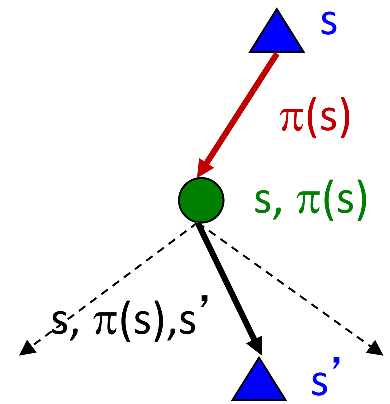
If B and E both go to C under this policy, how can their values be different?

Why Not Use Policy Evaluation?

- Simplified Bellman updates calculate V for a fixed policy:
  - Each round, replace V with a one-step-look-ahead layer over V

$$V_0^{\pi}(s) = 0$$

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



  - This approach fully exploited the connections between the states
  - Unfortunately, we need T and R to do it!

- Key question: how can we do this update to V without knowing T and R?
  - In other words, how to we take a weighted average without knowing the weights?

Sample-Based Policy Evaluation?

- We want to improve our estimate of V by computing these averages:

$$V^{\pi}_{k+1}(s) \leftarrow \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma V^{\pi}_{k}(s')]$$

- Idea: Take samples of outcomes s' (by doing the action!) and average

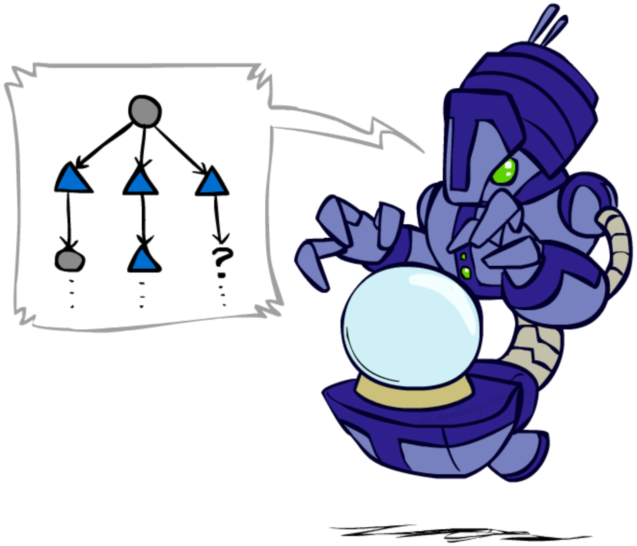
$$sample_1 = R(s,\pi(s),s'_1) + \gamma V^{\pi}_{k}(s'_1)$$

$$sample_2 = R(s,\pi(s),s'_2) + \gamma V^{\pi}_{k}(s'_2)$$

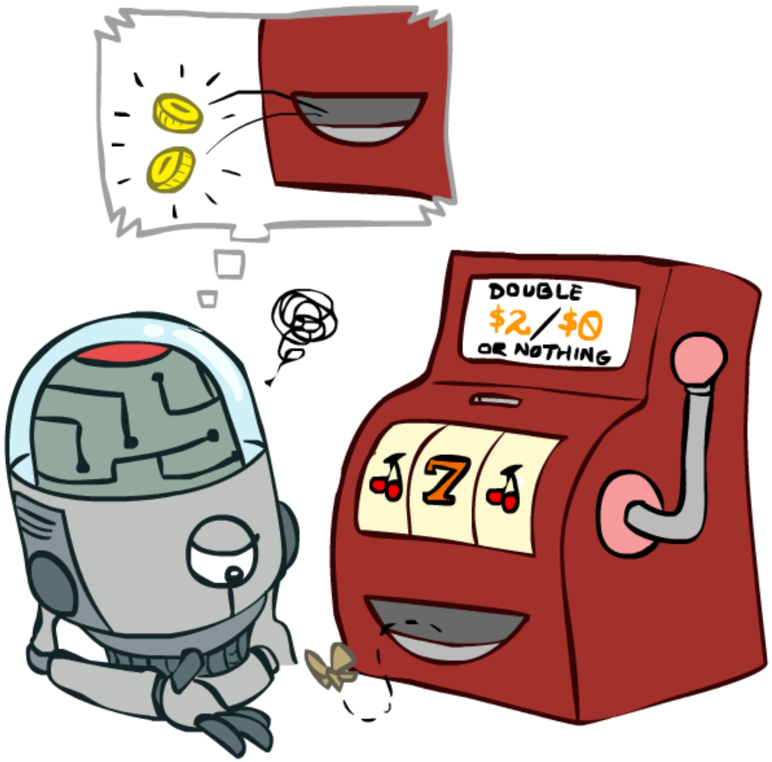
$$\ldots$$

$$sample_n = R(s,\pi(s),s'_n) + \gamma V^{\pi}_{k}(s'_n)$$

$$V^{\pi}_{k+1}(s) \leftarrow \frac{1}{n}\sum_{i} sample_i$$

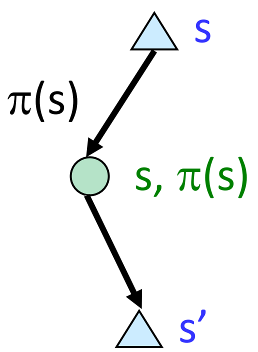


Temporal Difference (TD) Learning



Temporal Difference Learning

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



- Temporal difference learning of values
  - Policy still fixed, still doing evaluation!
  - 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): $$V^{\pi}(s) \leftarrow (1 - \alpha) V^{\pi}(s) + (\alpha) sample$$

Same update: $$V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha (sample - V^{\pi}(s))$$

Exponential Moving Average

- Exponential moving average
  - The running interpolation update: $\bar{x}_n = (1-\alpha) \cdot \bar{x}_{n-1} + \alpha \cdot x_n$
  - Makes recent samples more important:

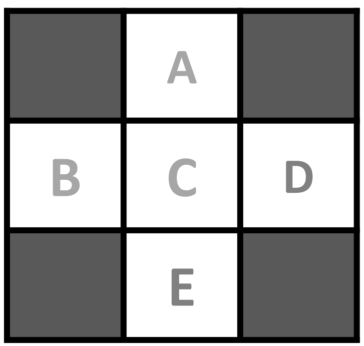
$$\bar{x}_n = \frac{x_n + (1-\alpha) \cdot x_{n-1} + (1-\alpha)^2 \cdot x_{n-2} + \ldots}{1 + (1-\alpha) + (1-\alpha)^2 + \ldots}$$

  - Forgets about the past (distant past values were wrong anyway)

- Decreasing learning rate (alpha) can give converging averages

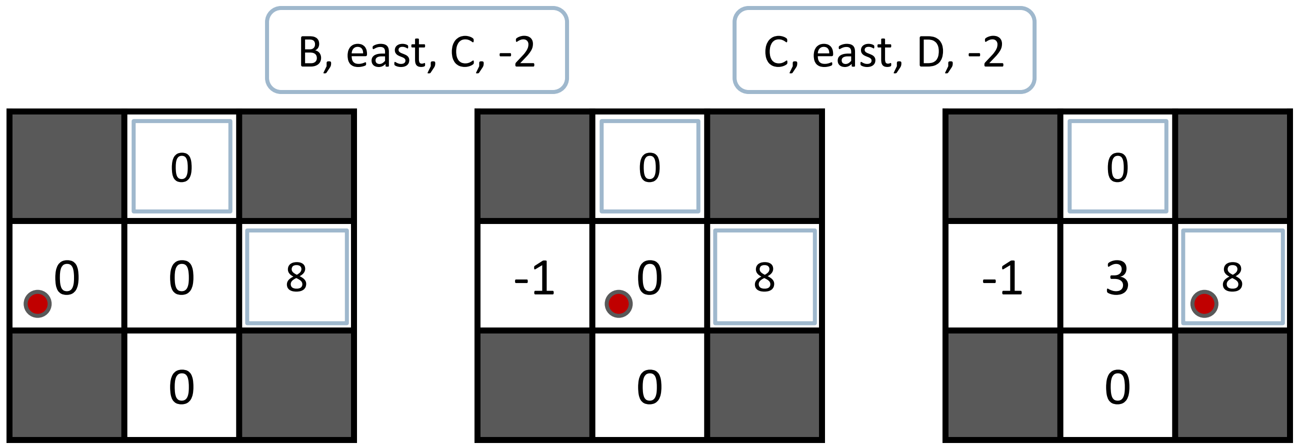
Example: Temporal Difference Learning

States



Assume: $\gamma = 1$, $\alpha = 1/2$

Observed Transitions



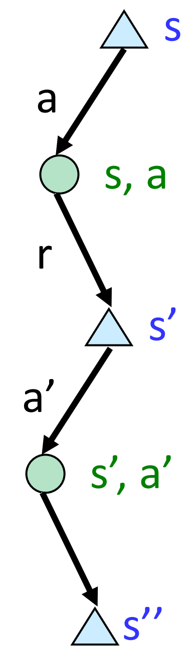
$$V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + \alpha\left[R(s,\pi(s),s') + \gamma V^{\pi}(s')\right]$$

Model-Free Learning

- Model-free (temporal difference) learning
  - Experience world through episodes

    $$(s, a, r, s', a', r', s'', a'', r'', s'''' \ldots)$$

  - Update estimates each transition (s, a, r, s')
  - Over time, updates will mimic Bellman updates



The Story So Far: MDPs and RL

Known MDP: Offline Solution

Goal	Technique
Compute V*, Q*, π*	Value / policy iteration
Evaluate a fixed policy π	Policy evaluation

Unknown MDP: Model-Based

Goal	Technique
Compute V*, Q*, π*	VI/PI on approx. MDP
Evaluate a fixed policy π	PE on approx. MDP

Unknown MDP: Model-Free

Goal	Technique
Compute V*, Q*, π*	Q-learning
Evaluate a fixed policy π	Value Learning

Detour: Q-Value Iteration

- Value iteration: find successive (depth-limited) values
  - Start with $V_0(s) = 0$, which we know is right
  - Given V_k, calculate the depth k+1 values for all states:

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s')\right]$$

- But Q-values are more useful, so compute them instead
  - Start with $Q_0(s,a) = 0$, which we know is right
  - Given Q_k, calculate the depth k+1 q-values for all q-states:

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')\right]$$

Problems with TD Value Learning

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

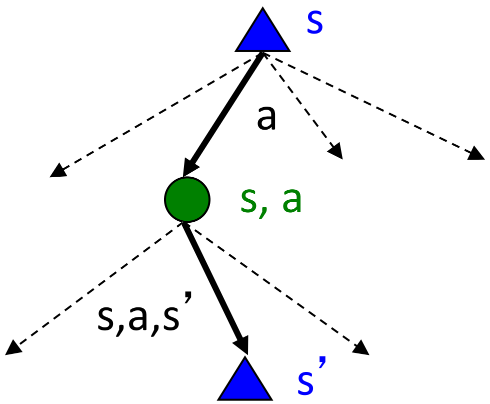
$$\pi^*(s) = \arg\max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

- Idea: learn Q-values, not values

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

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

- Makes action selection model-free too!



Approximating Values through Samples

- Policy Evaluation:

$$V^{\pi}_{k+1}(s) \leftarrow \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma V^{\pi}_{k}(s')]$$

- Value Iteration:

$$V_{k+1}(s) \leftarrow \max_{a} \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma\, V_k(s')\right]$$

- Q-Value Iteration:

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')\right]$$

Q-Learning

- Q-Learning: sample-based Q-value iteration

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a') \right]$$

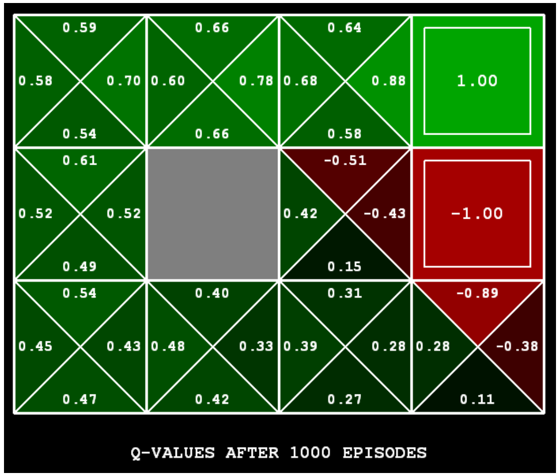
  - But can't compute this update without knowing T, R

- Learn Q(s,a) values as you go
  - Receive a sample (s,a,s',r)
  - Consider your old estimate: $Q(s,a)$
  - Consider your new sample estimate:

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

  - Incorporate the new estimate into a running average:

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



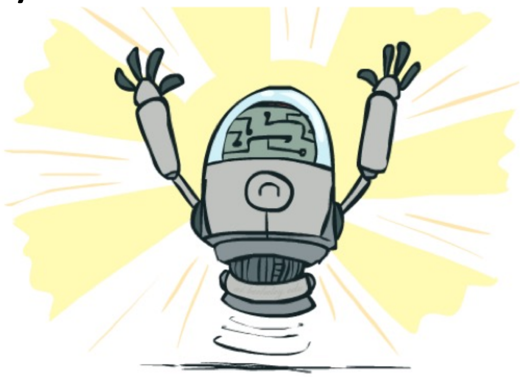
no longer
policy evaluation!

Video of Demo Q-Learning -- Gridworld

Video of Demo Q-Learning -- Crawler

Q-Learning Properties

- Amazing result: Q-learning converges to optimal policy -- even if samples are generated from a suboptimal policy!



- This is called off-policy learning

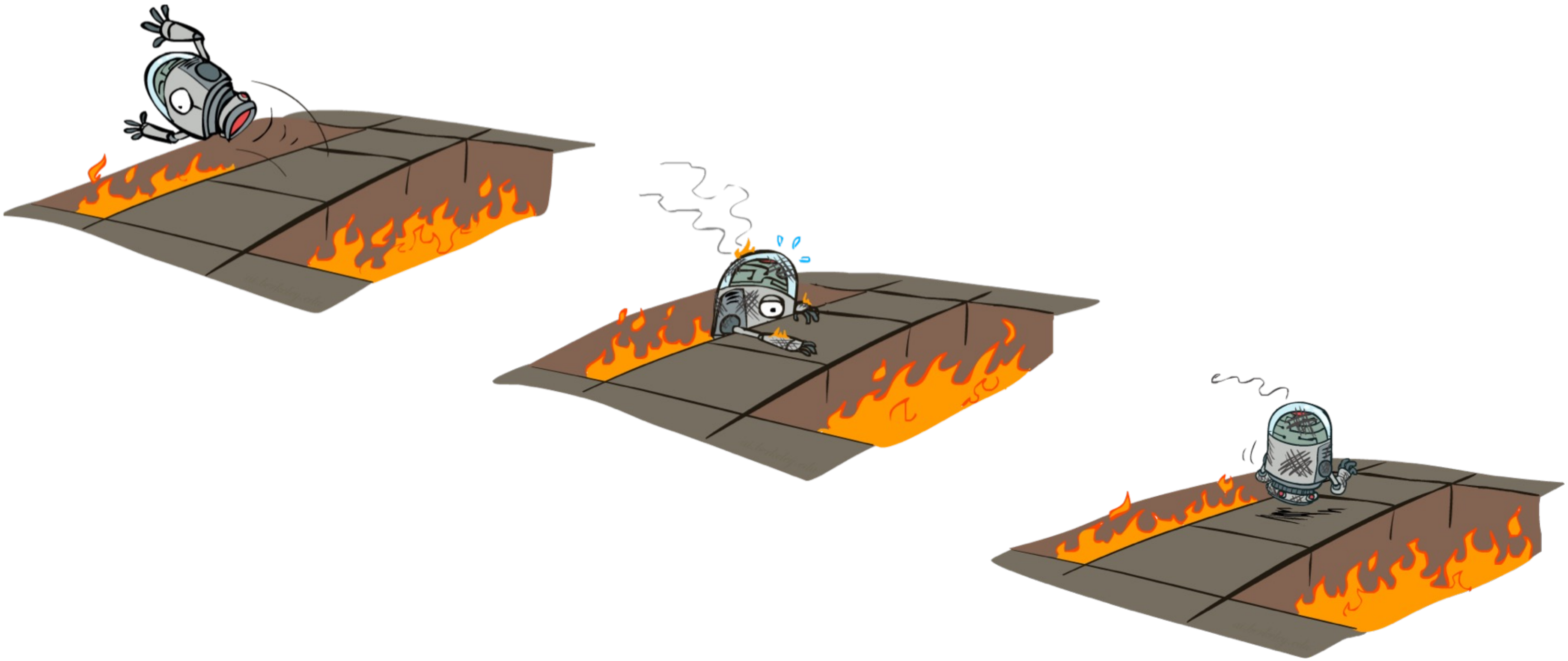
- Caveats:
  - You have to explore enough (eventually try every state/action pair infinitely often)
  - You have to decrease the learning rate appropriately
  - Basically, in the limit, it doesn't matter how you select actions (!)

Summary

- RL solves MDPs via direct experience of transitions and rewards
- There are several schemes:
  - Learn the MDP model and solve it
  - Learn V directly from sums of rewards, or by TD local adjustments
    - Still need a model to make decisions by lookahead
  - Learn Q by local Q-learning adjustments, use it directly to pick actions

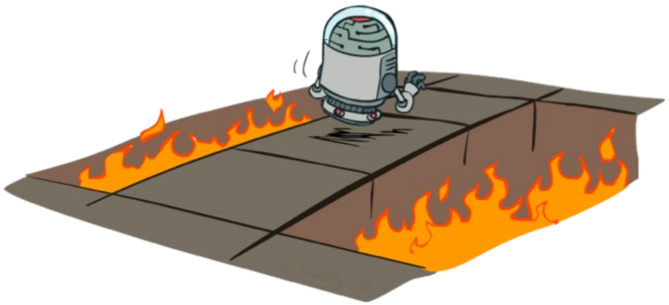
- Big missing pieces:
  - How to explore without too much regret?
  - How to scale this up to Tetris (10^{60}), Go (10^{172}), StarCraft ($|A|=10^{26}$)?

Active RL



Active RL

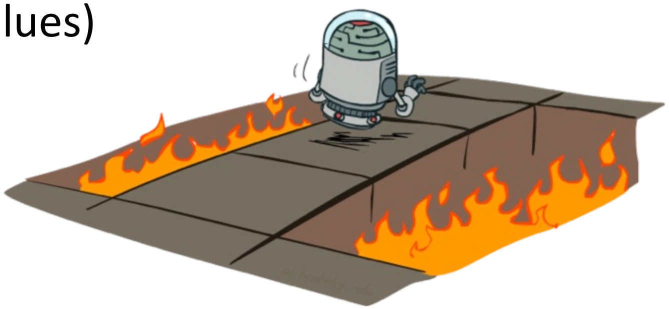
- Full reinforcement learning: optimal policies (like value iteration)
  - You don't know the transitions T(s,a,s')
  - You don't know the rewards R(s,a,s')
  - You choose the actions now
  - Goal: learn the optimal policy / values



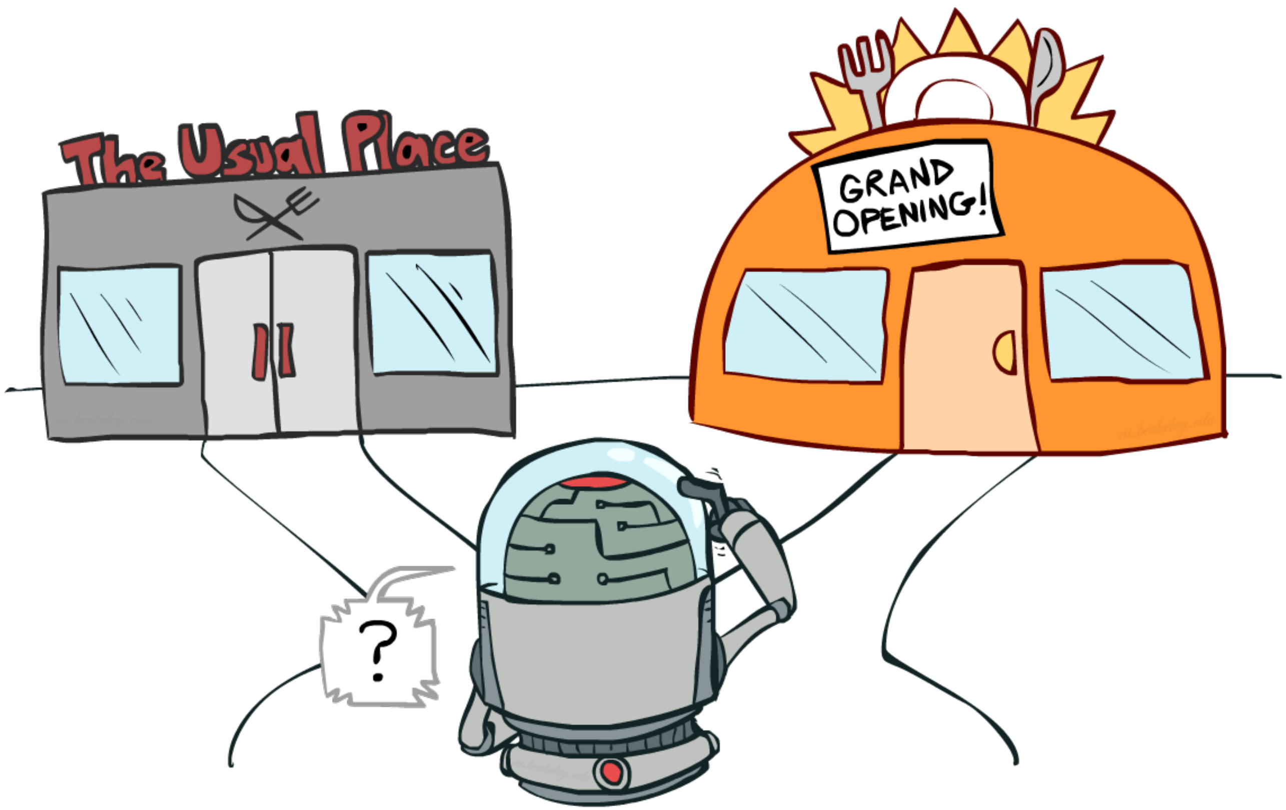
- In this case:
  - Learner makes choices!
  - Fundamental tradeoff: exploration vs. exploitation
  - This is NOT offline planning! You actually take actions in the world and find out what happens…

Model-Free Learning

- act according to current optimal (based on Q-Values)
- but also explore…



Exploration vs. Exploitation



Exploration vs exploitation

- ***Exploration***: try new things
- ***Exploitation***: do what's best given what you've learned so far
- Key point: pure exploitation often gets ***stuck in a rut*** and never finds an optimal policy!

Exploration method 1: ε-greedy

- ε-greedy exploration
  - Every time step, flip a biased coin
  - With (small) probability ε, act randomly
  - With (large) probability 1-ε, act on current policy

- Properties of ε-greedy exploration
  - Every s,a pair is tried infinitely often
  - Does a lot of stupid things
    - Jumping off a cliff *lots of times* to make sure it hurts
  - Keeps doing stupid things for ever
    - Decay ε towards 0

Video of Demo Q-learning – Manual Exploration – Bridge Grid

Q-learning: Policy

- Greedy action selection:

$$\pi(s) = \underset{a}{\operatorname{argmax}}\, Q(s,a)$$

- ϵ-greedy: greedy most of the times, occasionally take a random action

- Softmax policy: Give a higher probability to the actions that currently have better utility, e.g,

$$\pi(s,a) = \frac{b^{Q(s,a)}}{\sum_{a'} b^{Q(s,a')}}$$

- After learning Q^*, the policy is greedy?

Q-learning Algorithm

Initialize $Q(s, a)$ arbitrarily

Repeat (for each episode):

  Initialize s

  Repeat (for each step of episode):

    Choose a from s using a policy derived from Q (e.g., ε-greedy, softmax, …)

    Take action a, receive reward r, observe new state s'

    $Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right]$

    $s \leftarrow s'$

  until s is terminal

Q-learning convergence

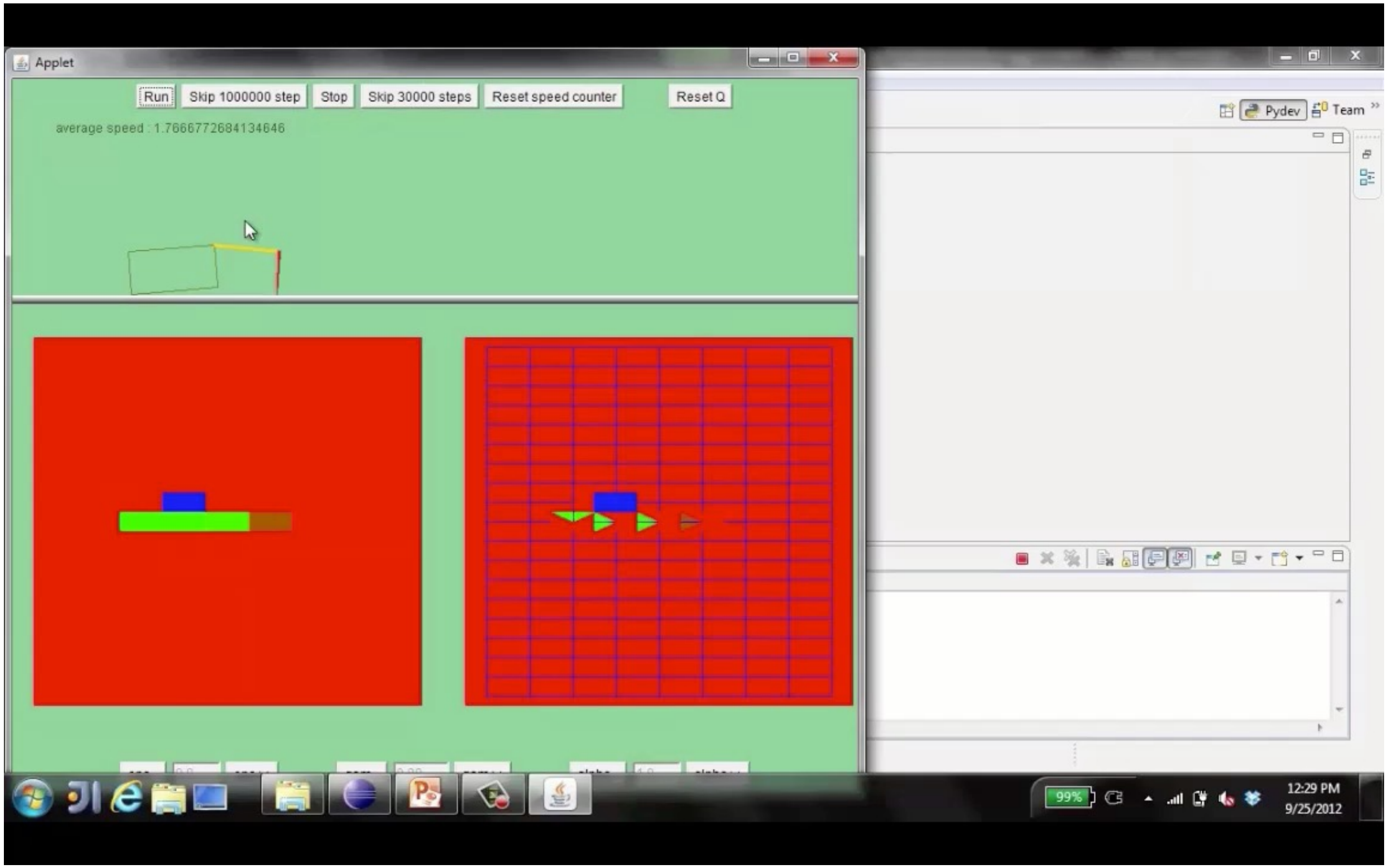
- Q-learning converges to optimal Q-values if
  - Every state is visited infinitely often
  - The policy for action selection becomes greedy as time approaches infinity
  - The step size parameter is chosen appropriately

- Stochastic approximation conditions
  - The learning rate is decreased fast enough but not too fast

Video of Demo Q-Learning Auto Cliff Grid

Video of Demo Q-learning – Epsilon-Greedy – Crawler

Video of Demo Q-Learning -- Crawler



Exploration Functions

- When to explore?
  - Random actions: explore a fixed amount
  - Better idea: explore areas whose badness is not (yet) established, eventually stop exploring

Exploration Functions

- Exploration function
  - Takes a value estimate u and a visit count n, and returns an optimistic utility, e.g.

$$f(u,n) = u + k/n$$

- Regular Q-update:
  - $Q(s,a) \leftarrow (1-\alpha)\cdot Q(s,a) \;+\; \alpha\cdot[R(s,a,s') + \gamma\, max_{a'} Q(s',a')\,]$
- Modified Q-update:
  - $Q(s,a) \leftarrow (1-\alpha)\cdot Q(s,a) \;+\; \alpha\cdot[R(s,a,s') + \gamma\, Q(s',a^e)\,]$

$$a^e = \text{argmax}_{a'} f(Q(s',a'), N(s',a'))$$

- Modified Q-update II:
  - $Q(s,a) \leftarrow (1-\alpha)\cdot Q(s,a) \;+\; \alpha\cdot[R(s,a,s') + \gamma \text{max}_{a'} f(Q(s',a'), N(s',a'))]$
  - Note: this propagates the "bonus" back to states that lead to unknown states as well!

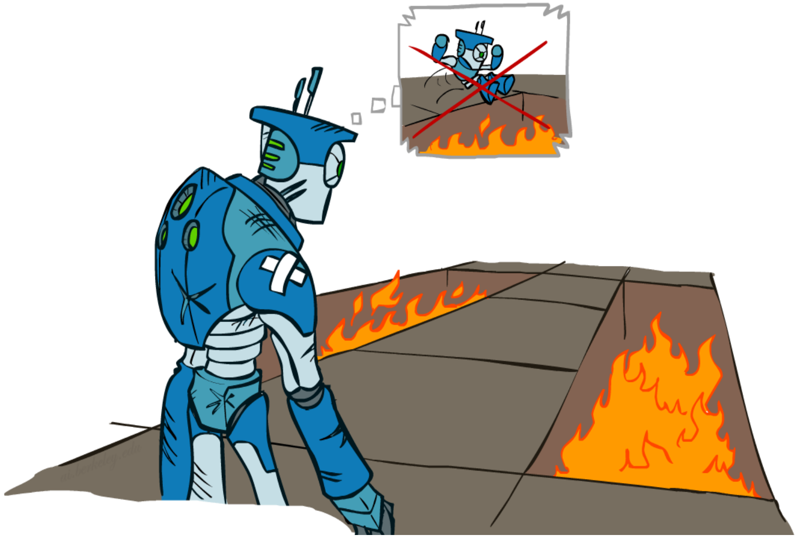
Video of Demo Q-learning – Exploration Function – Crawler

Regret

- Even if you learn the optimal policy, you still make mistakes along the way!

- Regret is a measure of your total mistake cost: the difference between your (expected) rewards, including youthful suboptimality, and optimal (expected) rewards

- Minimizing regret goes beyond learning to be optimal – it requires optimally learning to be optimal
  - Example: random exploration and exploration functions both end up optimal, but random exploration has higher regret



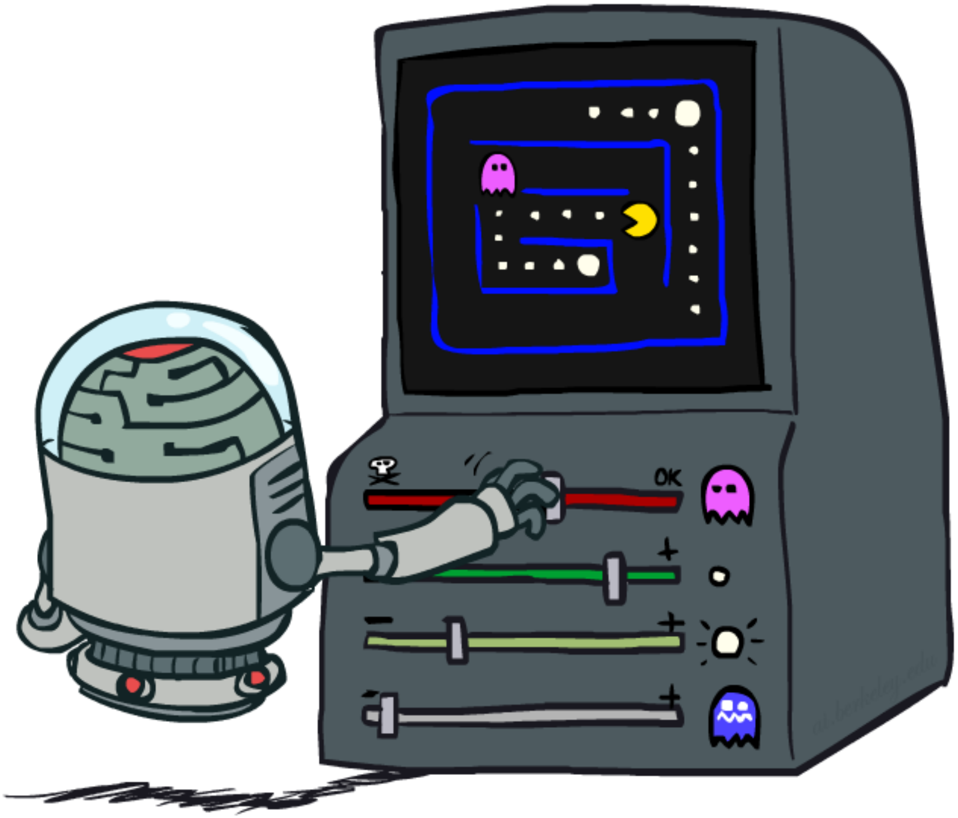
Tabular methods: Problem

- All of the introduced methods maintain a table

- Table size can be very large for complex environments
  - Too many states to visit them all in training
    - We may not even visit some states

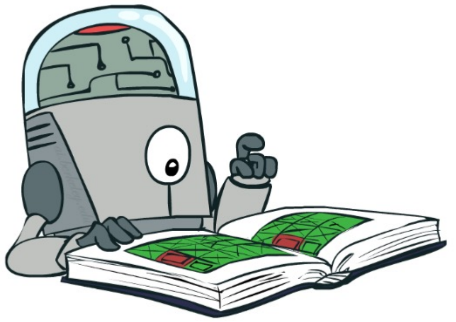
  - Too many states to hold the q-tables in memory
    - But computation and memory problem

Approximate Q-Learning



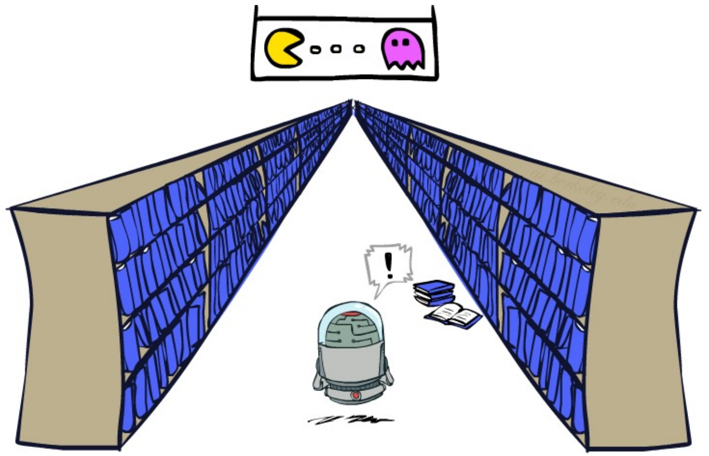
Generalizing Across States

- Basic Q-Learning keeps a table of all q-values



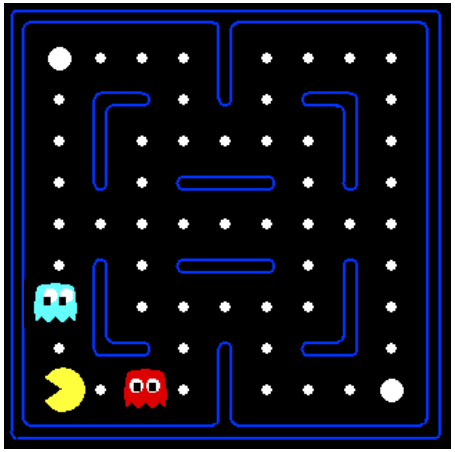
- In realistic situations, we cannot possibly learn about every single state!
  - Too many states to visit them all in training
  - Too many states to hold the q-tables in memory

- Instead, we want to generalize:
  - Learn about some small number of training states from experience
  - Generalize that experience to new, similar situations
  - This is a fundamental idea in machine learning, and we'll see it over and over again

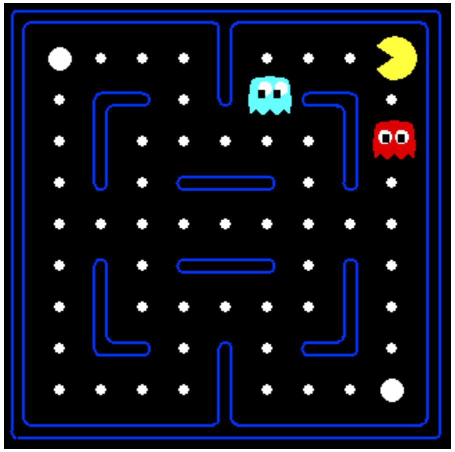


Example: Pacman

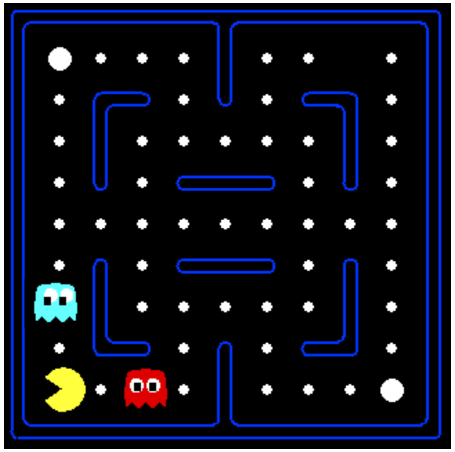
Let's say we discover through experience that this state is bad:



In naïve q-learning, we know nothing about this state:



Or even this one!



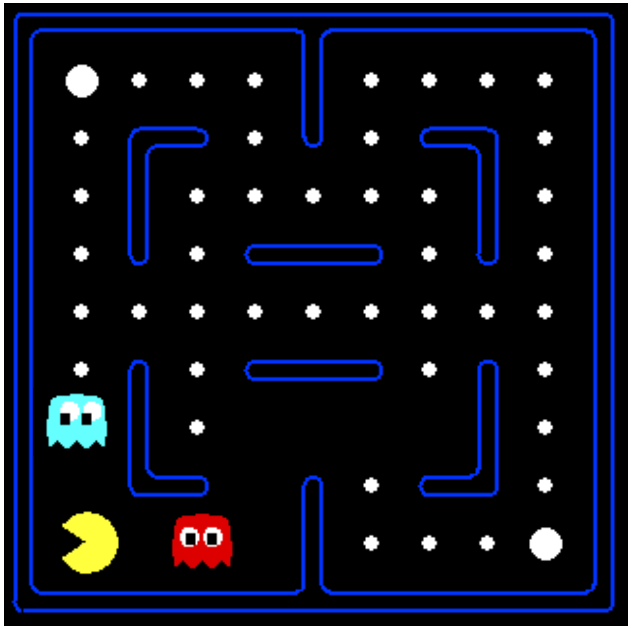
Video of Demo Q-Learning Pacman – Tiny – Watch All

Video of Demo Q-Learning Pacman – Tiny – Silent Train

Video of Demo Q-Learning Pacman – Tricky – Watch All

Feature-Based Representations

- Solution: describe a state using a vector of features (properties)
  - 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.
    - Is it the exact state on this slide?
  - Can also describe a q-state (s, a) with features (e.g. action moves closer to food)



Linear Value Functions

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

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

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

- With the wrong features, the best possible approximation may be terrible!

- But in practice we can compress a value function for chess (10^{43} states) down to about 30 weights and get decent play!!!

- Advantage: our experience is summed up in a few powerful numbers

- Disadvantage: states may share features but actually be very different in value!

Approximate Q-Learning

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

- Q-learning with linear Q-functions:

$$\text{transition} \;= (s, a, r, s')$$

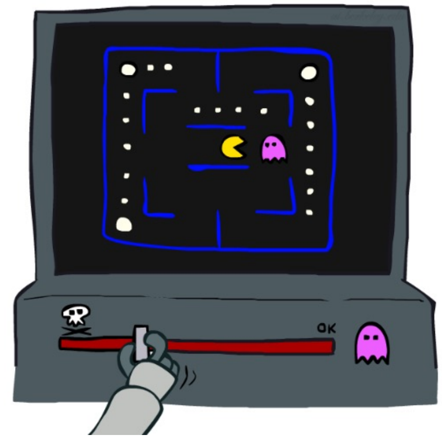
$$\text{difference} = \left[r + \gamma \max_{a'} Q(s', a') \right] - Q(s,a)$$

$$Q(s,a) \leftarrow Q(s,a) + \alpha \, [\text{difference}]$$

$$w_i \leftarrow w_i + \alpha \, [\text{difference}] \, f_i(s,a)$$

Exact Q's

Approximate Q's



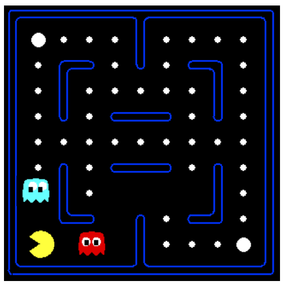
- Intuitive interpretation:
  - Adjust weights of active features
  - E.g., if something unexpectedly bad happens, blame the features that were on: disprefer all states with that state's features

- Formal justification: online least squares

Example: Q-Pacman

$$Q(s,a) = 4.0 f_{DOT}(s,a) - 1.0 f_{GST}(s,a)$$

s

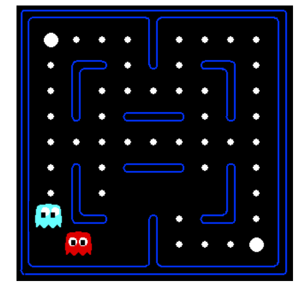


$f_{DOT}(s, \text{NORTH}) = 0.5$

$f_{GST}(s, \text{NORTH}) = 1.0$

$a = \text{NORTH}$
$r = -500$

s'



$Q(s, \text{NORTH}) = +1$

$r + \gamma \max_{a'} Q(s', a') = -500 + 0$

$Q(s', \cdot) = 0$

difference $= -501$

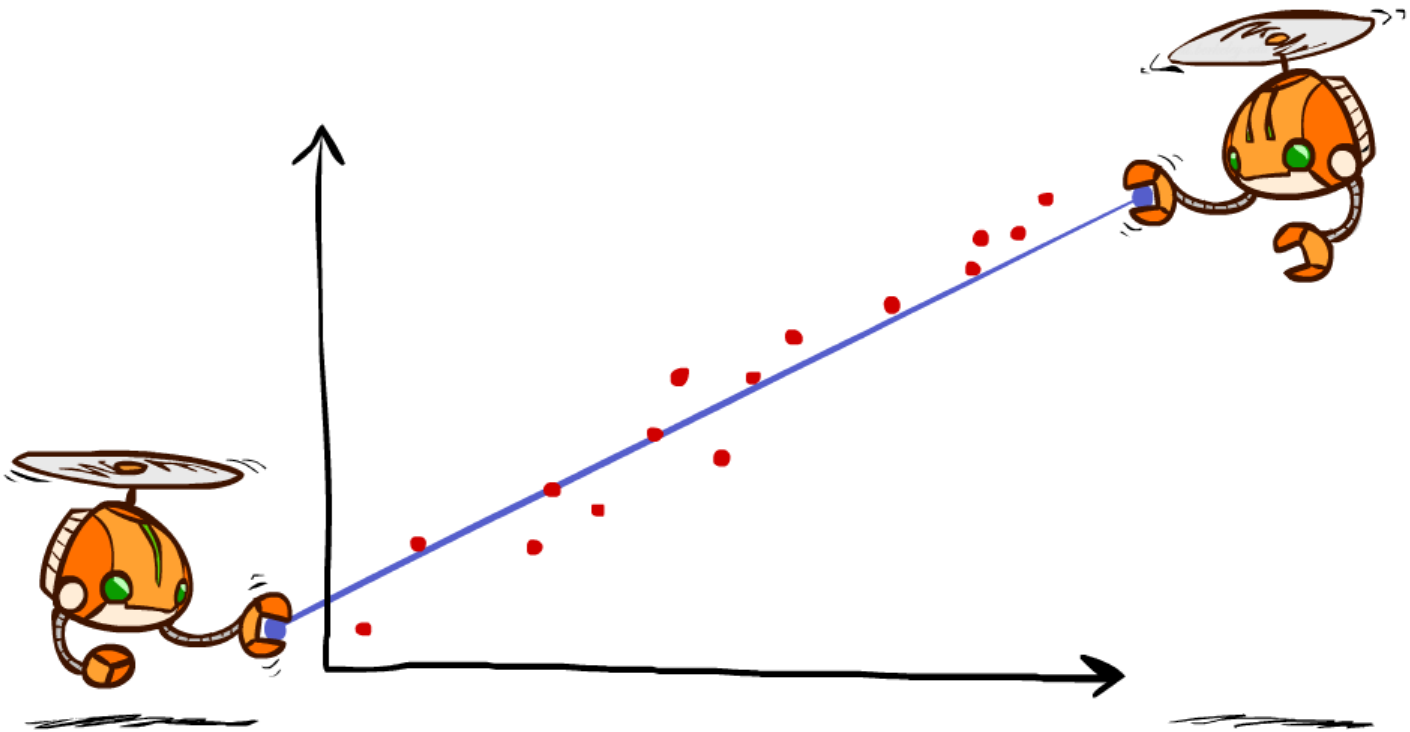
$w_{DOT} \leftarrow 4.0 + \alpha\,[-501]\,0.5$

$w_{GST} \leftarrow -1.0 + \alpha\,[-501]\,1.0$

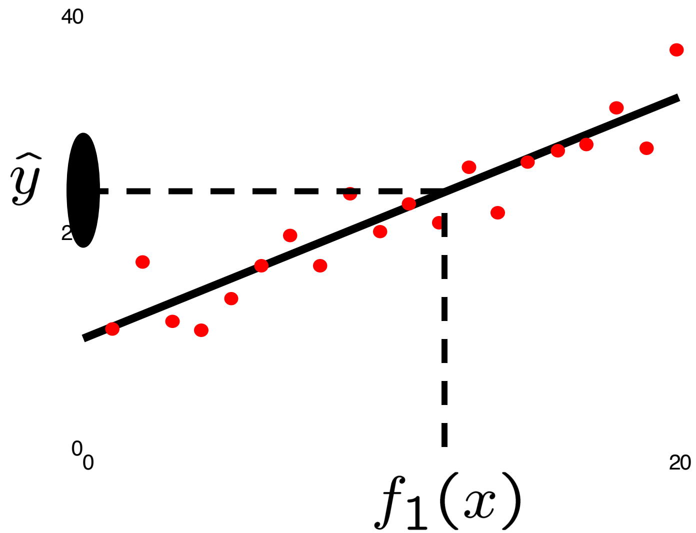
$$Q(s,a) = 3.0 f_{DOT}(s,a) - 3.0 f_{GST}(s,a)$$

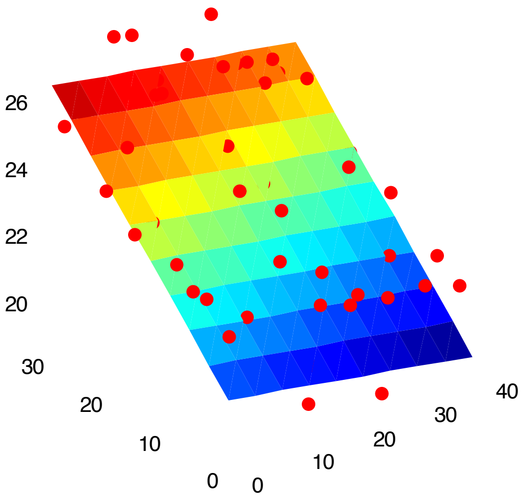
Video of Demo Approximate Q-Learning -- Pacman

Q-Learning and Least Squares



Linear Approximation: Regression*



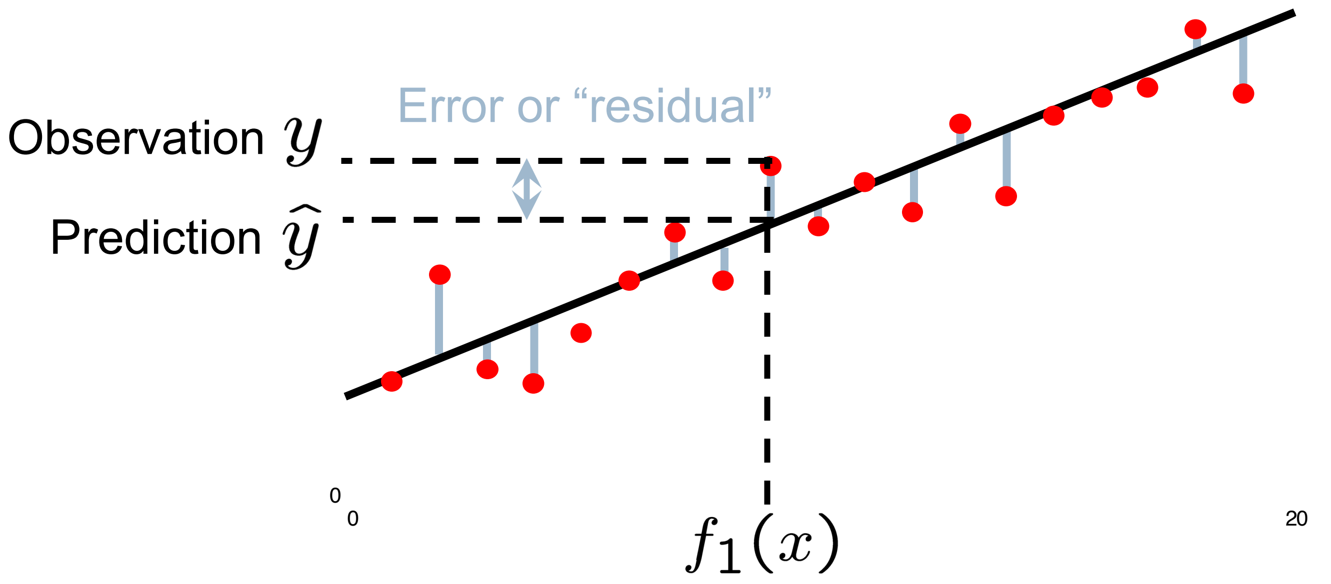


Prediction:
$\hat{y} = w_0 + w_1 f_1(x)$

Prediction:
$\hat{y}_i = w_0 + w_1 f_1(x) + w_2 f_2(x)$

Optimization: Least Squares*

$$\text{total error} = \sum_i (y_i - \hat{y}_i)^2 = \sum_i \left(y_i - \sum_k w_k f_k(x_i) \right)^2$$



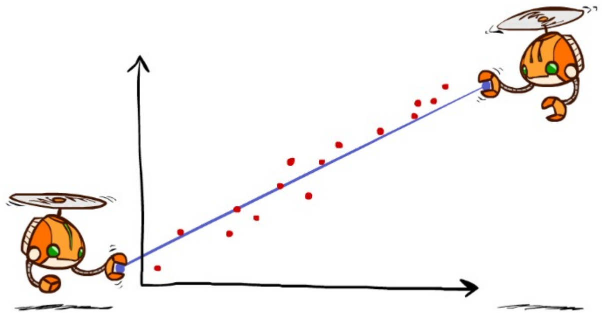
Minimizing Error*

Imagine we had only one point x, with features f(x), target value y, and weights w:

$$\text{error}(w) = \frac{1}{2}\left(y - \sum_k w_k f_k(x)\right)^2$$

$$\frac{\partial\ \text{error}(w)}{\partial w_m} = -\left(y - \sum_k w_k f_k(x)\right) f_m(x)$$

$$w_m \leftarrow w_m + \alpha \left(y - \sum_k w_k f_k(x)\right) f_m(x)$$

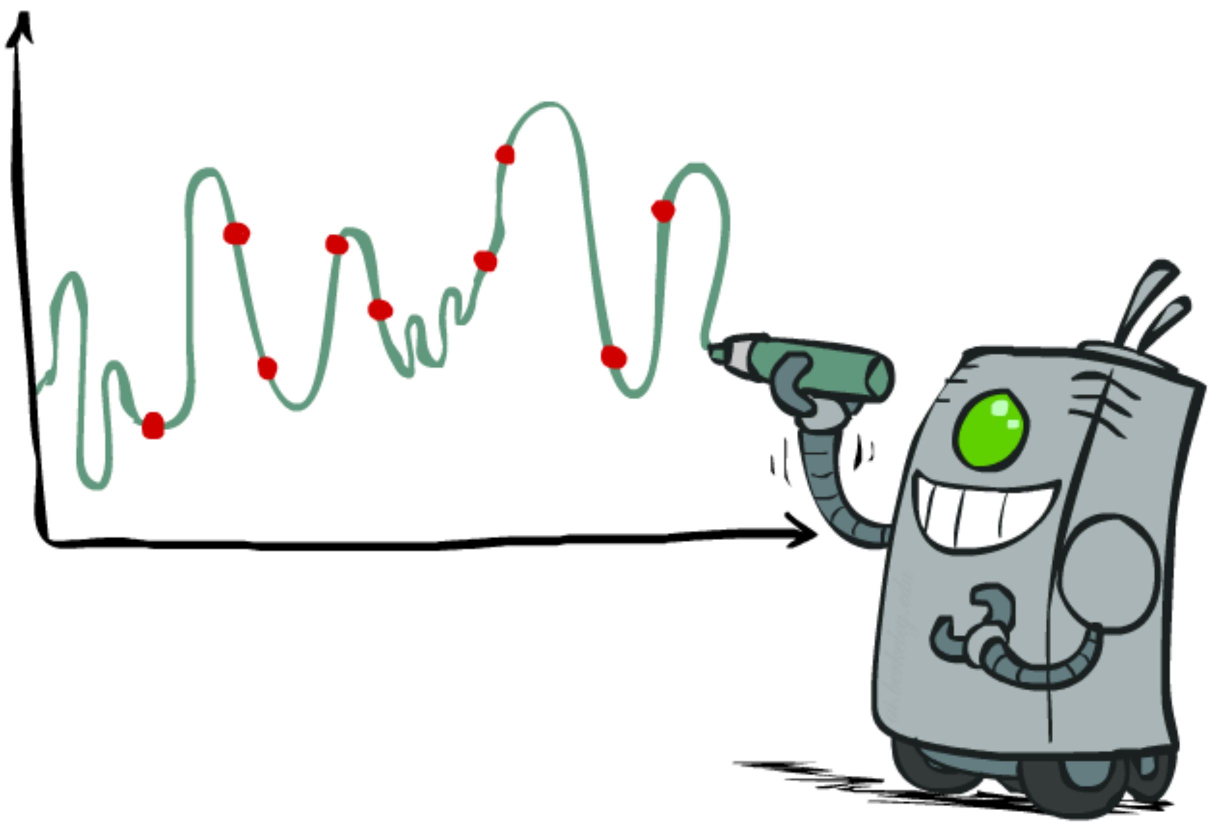


Approximate q update explained:

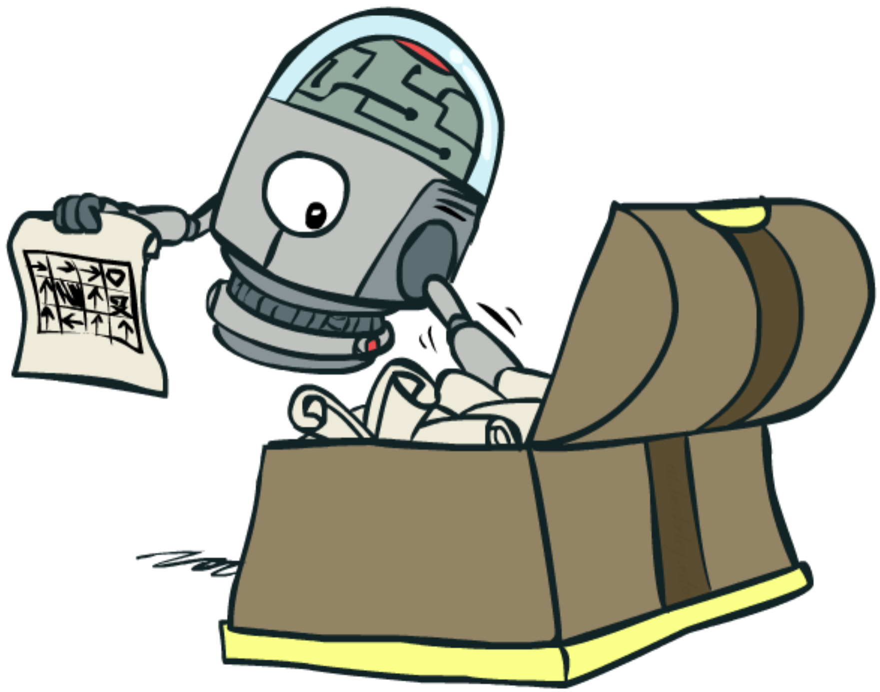
$$w_m \leftarrow w_m + \alpha \left[r + \gamma \max_a Q(s', a') - Q(s, a)\right] f_m(s, a)$$

"target" "prediction"

Overfitting: Why Limiting Capacity Can Help*



Policy Search



Policy Search

- Problem: often the feature-based policies that work well (win games, maximize utilities) aren't the ones that approximate V / Q best
  - Q-learning's priority: get Q-values close (modeling)
  - Action selection priority: get ordering of Q-values right (prediction)
  - We'll see this distinction between modeling and prediction again later in the course

- Solution: learn policies that maximize rewards, not the values that predict them

- Policy search: start with an ok solution (e.g. Q-learning) then fine-tune by hill climbing on feature weights

Policy Search

- Simplest policy search:
  - Start with an initial linear value function or Q-function
  - Nudge each feature weight up and down and see if your policy is better than before

- Problems:
  - How do we tell the policy got better?
  - Need to run many sample episodes!
  - If there are a lot of features, this can be impractical

- Better methods exploit lookahead structure, sample wisely, change multiple parameters…

Example from Pieter Abbeel

The Story So Far: MDPs and RL

Known MDP: Offline Solution

Goal	Technique
Compute V*, Q*, π*	Value / policy iteration
Evaluate a fixed policy π	Policy evaluation

Unknown MDP: Model-Based

Goal	**use features to generalize*	Technique
Compute V*, Q*, π*		VI/PI on approx. MDP
Evaluate a fixed policy π		PE on approx. MDP

Unknown MDP: Model-Free

Goal	**use features to generalize*	Technique
Compute V*, Q*, π*		Q-learning
Evaluate a fixed policy π		Value Learning

Summary

- Exploration vs. exploitation
  - Exploration guided by unfamiliarity and potential
  - Appropriately designed bonuses tend to minimize regret

- Generalization allows RL to scale up to real problems
  - Represent V or Q with parameterized functions
  - Adjust parameters to reduce sample prediction error

Conclusion

- We're done with Part I: Search and Planning!
- We've seen how AI methods can solve problems in:
  - Search
  - Constraint Satisfaction Problems
  - Games
  - Markov Decision Problems
  - Reinforcement Learning
- Next up: Part II: Reasoning, Uncertainty and Learning!