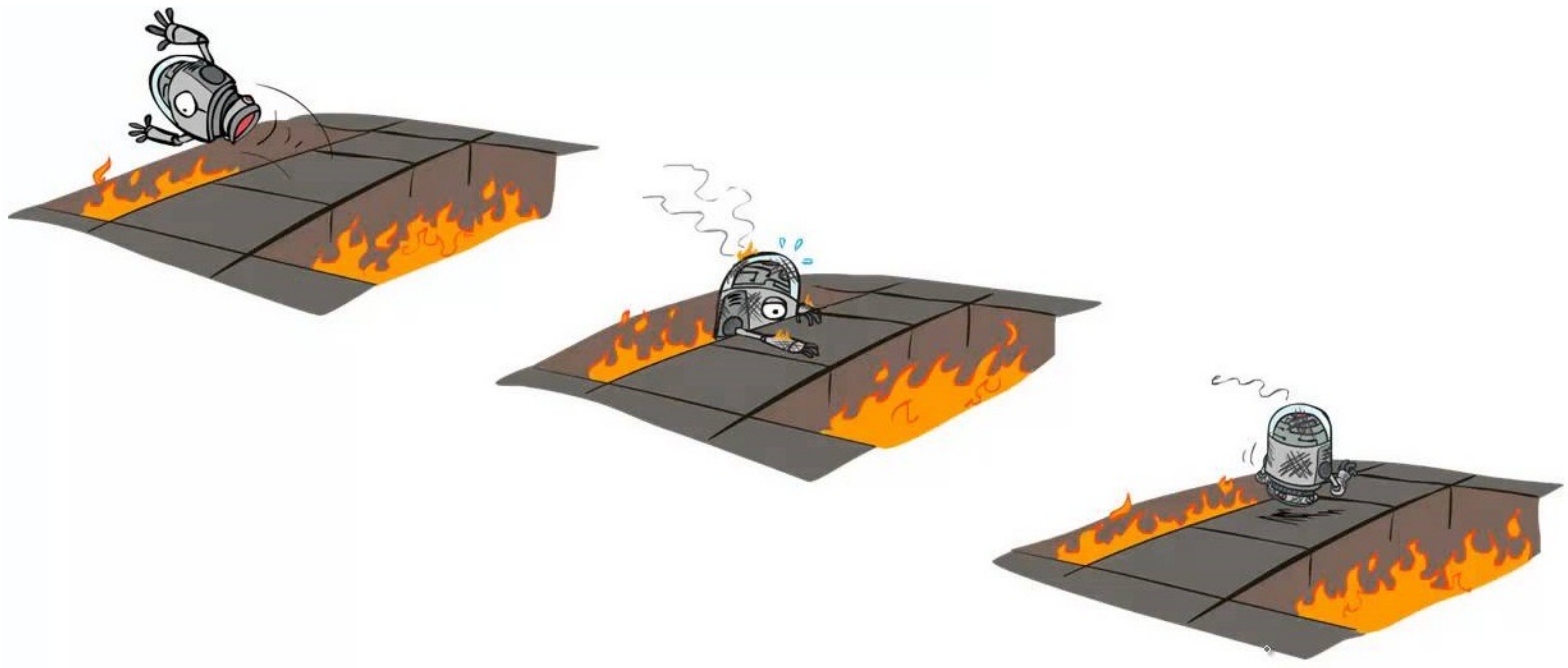
COMP 341 Intro to AI
Reinforcement Learning

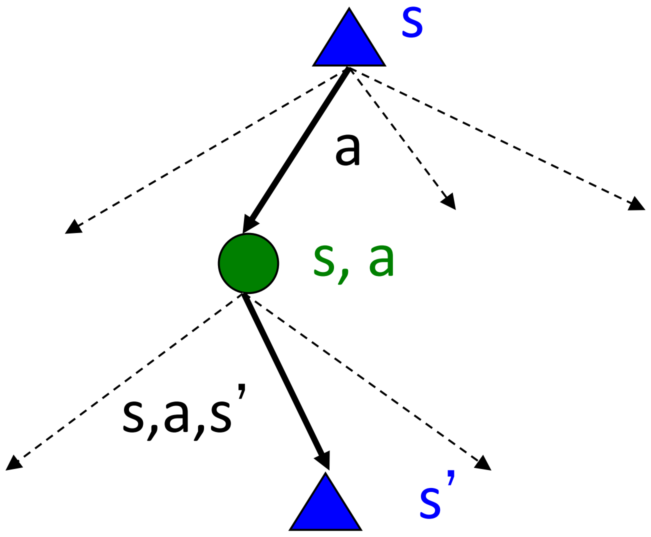


Asst. Prof. Barış Akgün

Koç University

MDPs

- Markov decision processes:
  - Set of states S
  - Set of actions A
  - Rewards R(s,a,s') (and discount γ)
  - Transition Model (P(s'|s,a) or T(s,a,s'))
  - Start state distribution
  - Sometimes terminal states



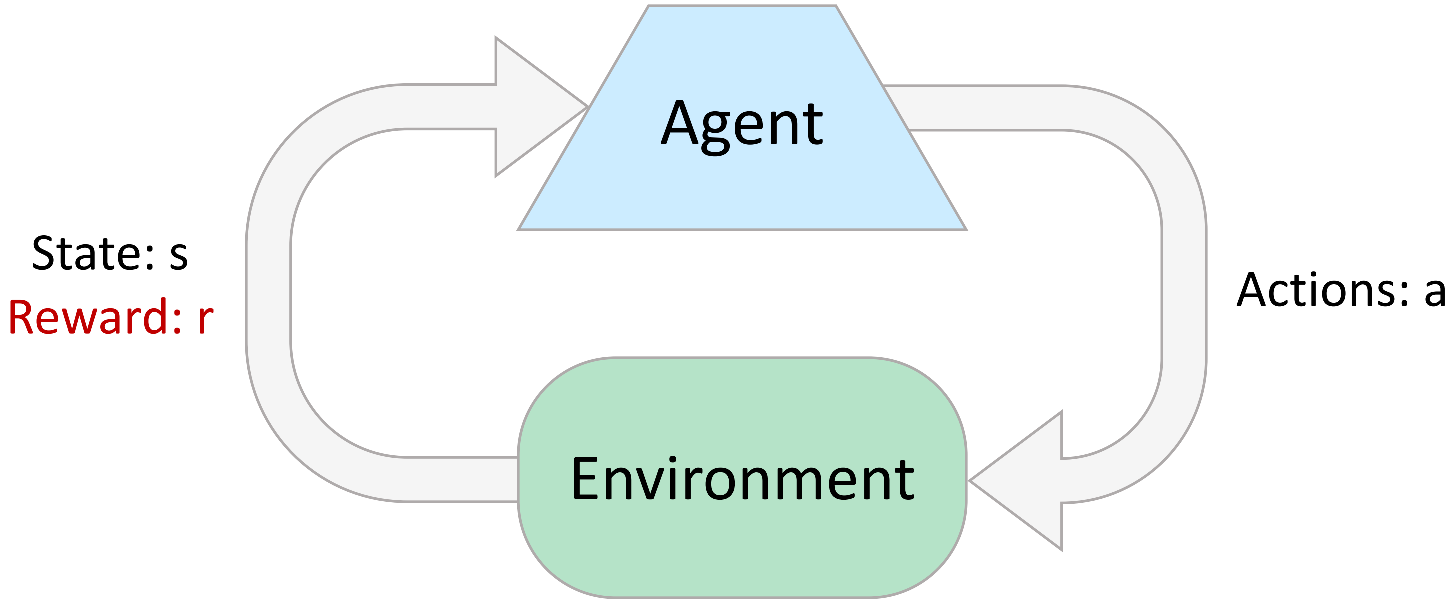
- Quantities:
  - Value function
  - Policies
  - Q-values (value for a state action pair)

Calculating Policies

- Goal is to find an optimal policy given the MDP
- Methods:
  - Value Iteration
  - Policy Iteration

- All the calculations are offline!
- Doing this online is called Reinforcement Learning

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!

Learning to Walk

Initial

Final

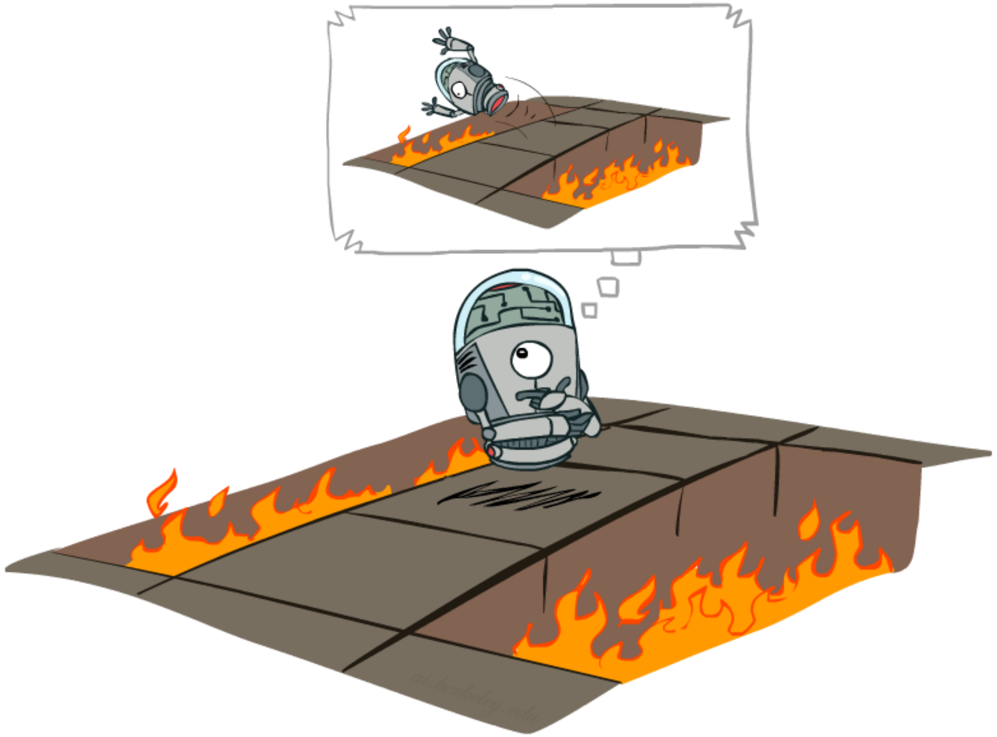
[Kohl and Stone, ICRA 2004]

Reinforcement Learning

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

- New twist: don’t know T or R (only receive rewards online) or both
  - i.e. we don’t know which states are good or what the actions do
  - Must actually try actions and states out to learn

Offline (MDPs) vs. Online (RL)



Offline Solution

Online Learning

Why Reinforcement Learning?

- It’s not that we would prefer to “risk” the agent, but we are often not left a choice
    - Models cannot be approximated reliably (e.g. a spider robot walking on snow)
    - Reward may be unknown (e.g. showing ads to maximize profit)
- It turns out that reinforcement learning is the only feasible approach to train an agent to perform at high levels in complex and uncertain domains
- Also note that we often have either the transition model or the reward function (but not both)

- Even in some cases where we have the models, offline learning borrows some ideas from online methods
- In this case the learning happens online “in simulation”

Motivation

- In the agent way of thinking, rewards can be considered as a sensory input to the agent
- However, the agent must recognize the reward as reward!

- Animals are hardwired to recognize pain and hunger as bad reward and pleasure and food intake as positive reward

- Food can be used to teach tricks to animals!

Motivation

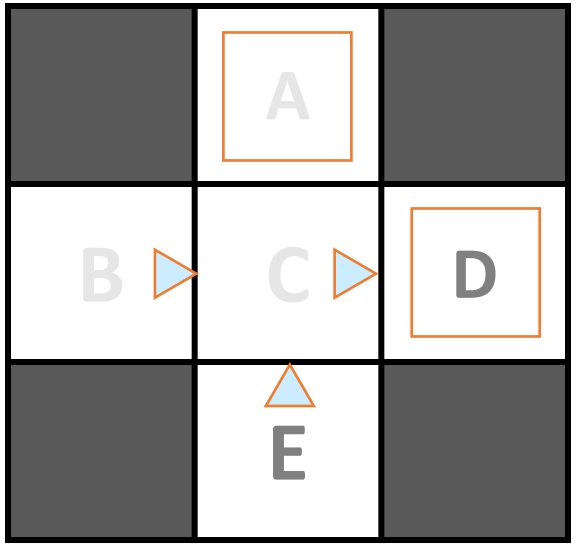
- Animals are not born with “physics” knowledge but somethings are built-in
- They learn how to move their bodies starting from innate knowledge
  - E.g., babies start from reflexes
- They re-learn if their body changes
  - E.g., teenagers’ voice cracks and they are clumsy growing up
  - E.g., when they are injured
- They also learn to manipulate objects through trial-error and rewards

Model-Based Learning

- Model-Based Idea:
  - Learn an approximate model based on experiences (supervised learning)
  - 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 $P(s'|s,a)$
  - Discover each $R(s,a,s')$ when we experience (s, a, s’)
- Step 2: Solve the learned MDP
  - We already know the methods!

Example: Model Based Learning

Input Policy π



Assume: $\gamma = 1$

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

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

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

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

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

Example: Expected Age

Goal: Compute expected age of a group of people

Known P(A)

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

Without P(A), instead collect samples $[a_1, a_2, \ldots a_N]$

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.

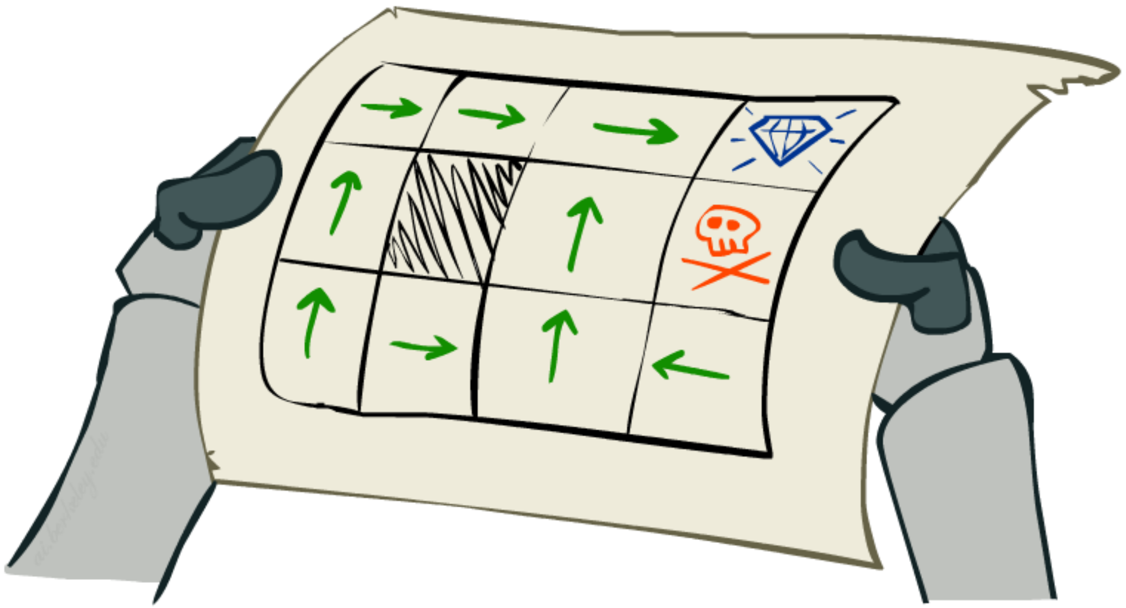
Model-Free Learning

- Our main goal is to learn a policy
- In both value iteration and policy iteration we learned the policy through calculating the values of the states
- It turns out that we can by-pass learning models and directly learn the values of states in an online fashion

- Model-Based vs Model-Free choice heavily depends on the problem, it is difficult to say in general which one is better
- We will have a discussion after learning more about RL

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



- 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 Utility Estimation

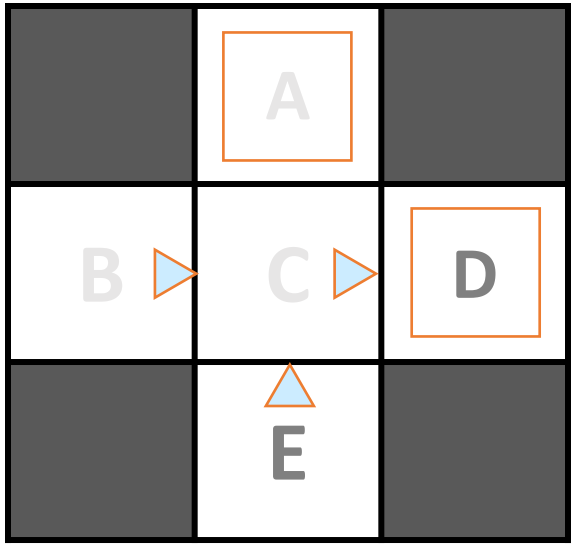
- Goal: Compute values for each state under π

- Idea: Average together observed sample values
  - Act according to π
  - For each visited state, write down what the sum of discounted rewards turned out to be
  - Average the sample values

- This is called direct evaluation

Example: Direct Evaluation

Input Policy π



Assume: $\gamma = 1$

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

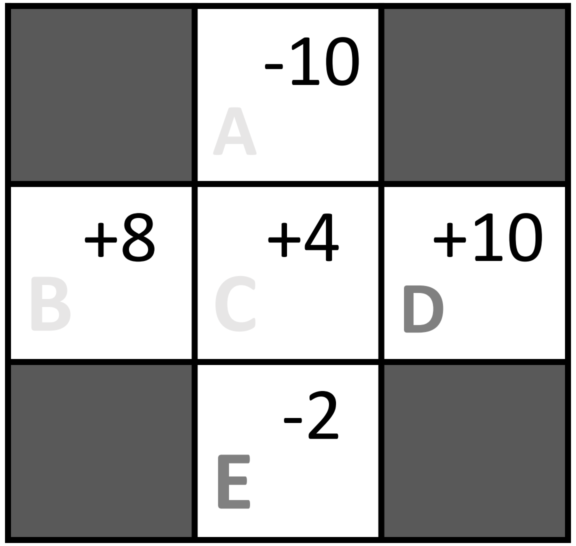
Episode 3

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

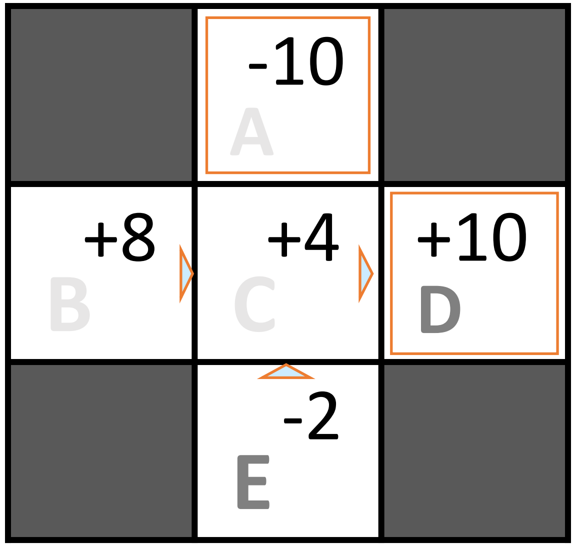


Problems with Direct Evaluation

- 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 bad about it?
  - It does not use information about state connections
  - Each state must be learned separately
  - So, it takes a long time to learn

Output Values



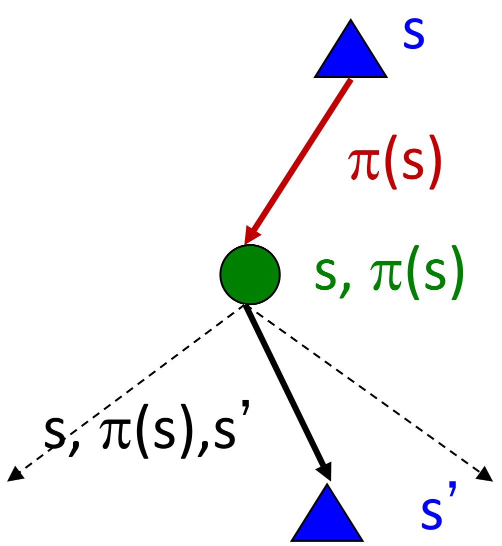
If B and E both go to C under this policy, how can their values be different?

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?

Model-Based Case

- Act according to a fixed policy
- Update the models with the observed episodes
- Do policy evaluation
  - Value iteration OR
  - Solving the linear system

- Called the “Passive Adaptive Dynamic Programming” in the AIMA book

- Note: Anything not labelled Model-Based is Model-Free in these slides

Sample Based Policy Evaluation

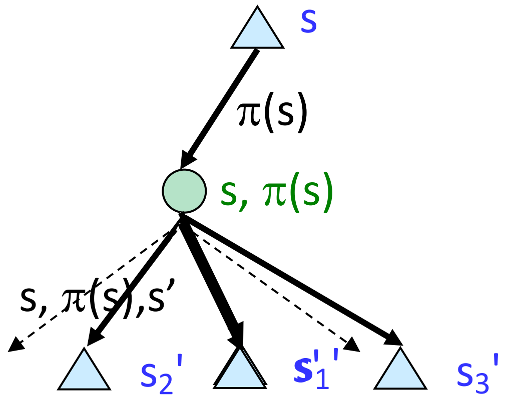
- We want to improve our estimate of V by computing these averages:

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

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

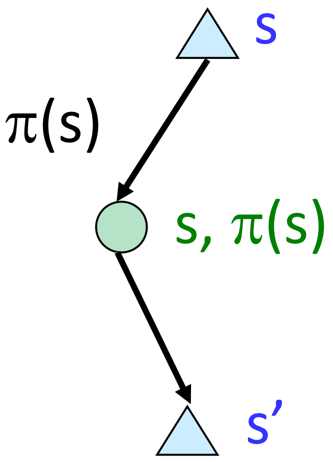
$$sample_1 = R(s, \pi(s), s_1') + \gamma V_k^{\pi}(s_1')$$
$$sample_2 = R(s, \pi(s), s_2') + \gamma V_k^{\pi}(s_2')$$
$$\ldots$$
$$sample_n = R(s, \pi(s), s_n') + \gamma V_k^{\pi}(s_n')$$

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



Almost! But we can’t rewind time to get sample after sample from state s.

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

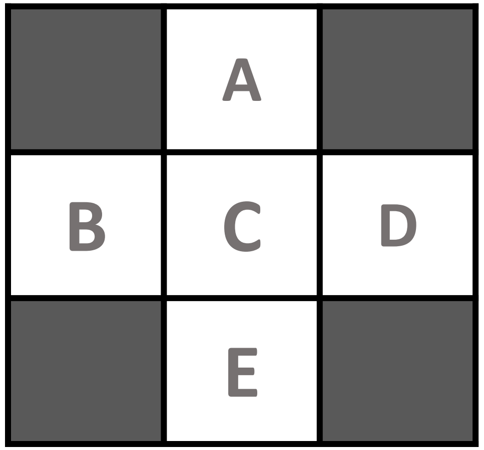
Update to V(s): $V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + (\alpha)sample$

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

- Values do not always converge with a fixed α, it needs to be decayed
- We need to decay it as $O(1/t)$, where t is the iteration number, but not too quickly!

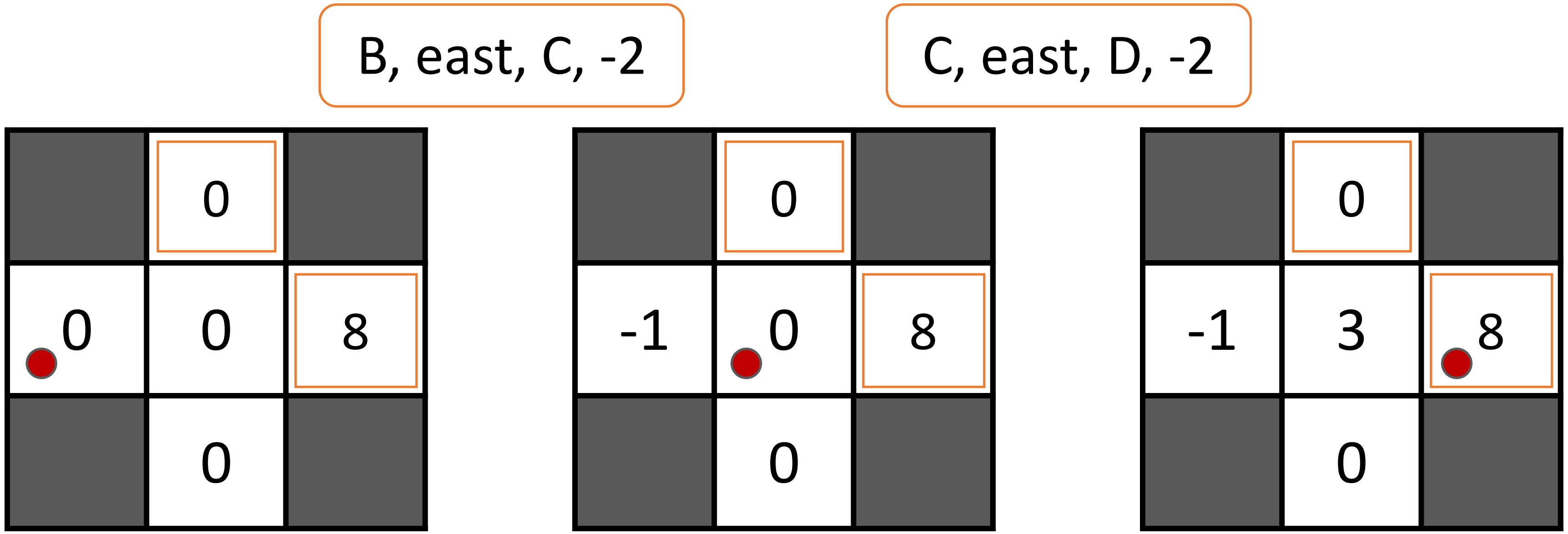
Example: TD Learning

States



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

Observed Transitions



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

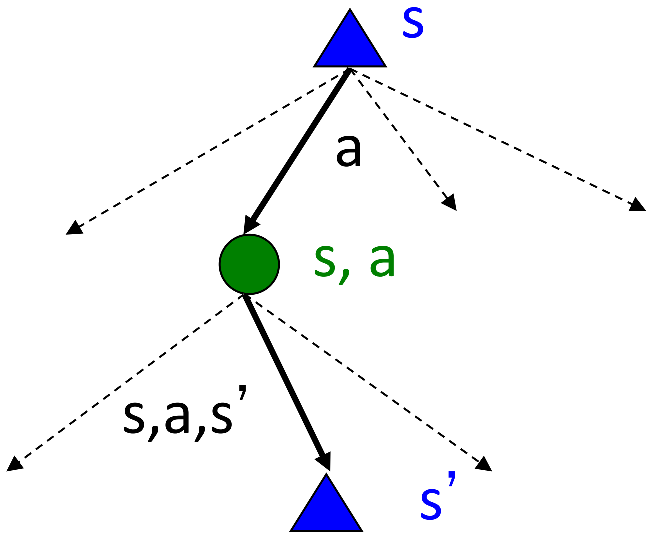
Problems with TD-Learning

- TD value learning is a model-free way to do policy evaluation, mimicking Bellman updates with running sample averages
- Once we have the values, can we get to actions? Can we do policy extraction?

$$\pi^*(s) = \arg_a \max(\sum_{s'} \boxed{(P(s'|s,a)\big(R(s,a,s')} + \gamma V^*(s')\big)))$$

- We cannot get a new policy in a model-free way!
- Idea: learn Q-values, not values
- Makes action selection model-free too!

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



Detour: Q-Values

- Value iteration: find successive values
  - Start with $V_0(s) = 0$
  - Given V_k, calculate the 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 for Model-Free RL, so compute them instead
  - Start with $Q_0(s,a) = 0$
  - Given Q_k, calculate the k+1 q-values for all q-states:

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

Q-Learning

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

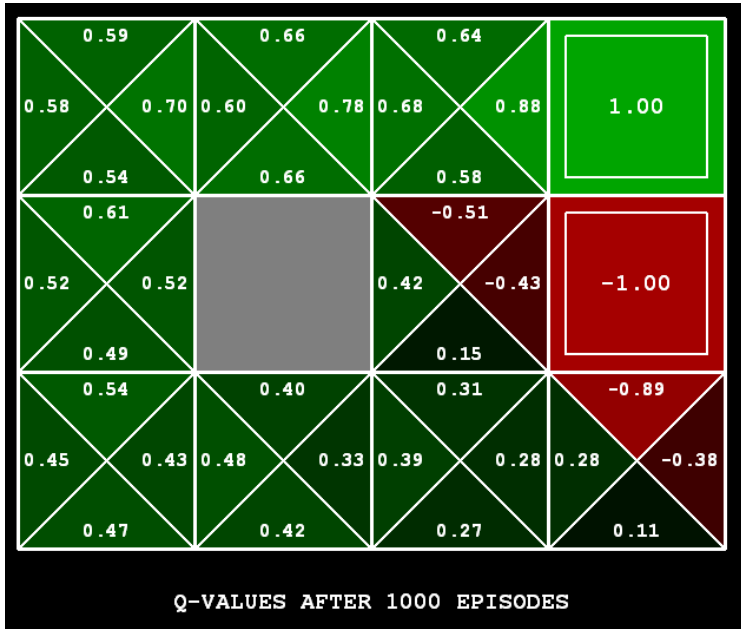
$$Q_{k+1}(s,a) = \sum_{s'} P(s'|s,a)\left(R(s,a,s') + \gamma \max_{a'} Q_k(s',a')\right)$$

- 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:

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



Active Reinforcement Learning

- So far the agent has been following a fixed policy
- Full reinforcement learning: optimal policies
  - You don't know the transitions T(s,a,s') / P(s'|s,a,)
  - 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! Which action to take?
  - Fundamental tradeoff: exploration vs. exploitation
  - This is NOT offline planning! You take actions in the world and find out what happens…

Model Based Version

- Learn a model and calculate the values using a fixed policy i.e. do passive RL
- Extract a new policy from the values
- Repeat
- Is this a good idea?
- No, it is not! Learning a model with this greedy strategy is a bad idea
  - Learned model will not be a good approximation of the real model
  - As a result, the extracted policy will be suboptimal in the real environment

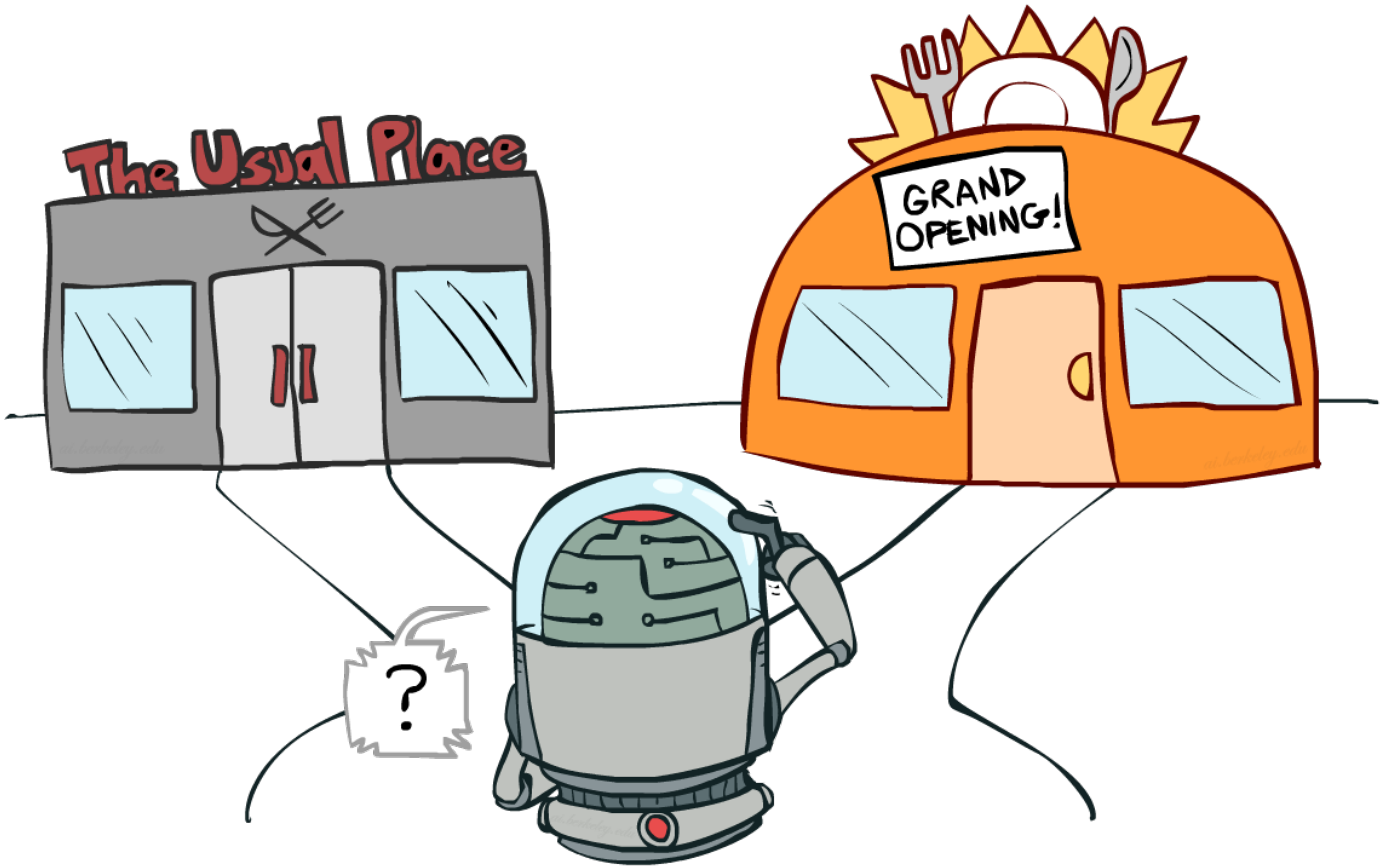
- This approach "**exploits**" what we already know
- We also want to "**explore**" as opposed to following the policies to get a better sense of the real model which will in turn let us get more reward!

What about Q-Learning?

- Same arguments go for Q-Learning, we need to “**explore**” enough to get good policies.
- Amazing result: Q-learning converges to optimal policy with enough exploration -- even if you’re acting sub-optimally!
- This is called off-policy learning

- Caveats:
  - You must explore enough
  - You must eventually make the learning rate small enough
    - … but not decrease it too quickly
  - Basically, in the limit, it doesn’t matter how you select actions (if there is exploration)

Exploration vs. Exploitation



How to Explore?

- Simplest: random actions (ε-greedy)
  - Every time step, flip a coin
  - With (small) probability ε, act randomly
  - With (large) probability 1-ε, act on current policy

- Problems with random actions?
  - You do eventually explore the space, but keep thrashing around once learning is done
  - One solution: lower ε over time
  - Another solution: exploration functions

How to Explore? – 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 function
  - Takes a value estimate u and a visit count n, and returns an optimistic utility, e.g.

$$f(u,n) = u + k/n \text{ or } f(u,n) \begin{cases} R^{+}, \text{if } n < N_e \\ u \text{ otherwise} \end{cases}$$

  - This makes the agent to think that there is high reward in the unexplored regions and forces it to explore those regions
- In essence, reward the unseen states more

Update Equations with Exploration Functions

- Value update (Model-based)

$$V(s) = \max_a \left[f\left(\underbrace{\boxed{\sum_{s'} \left(P(s'|s,a)\left(R(s,a,s') + \gamma V(s')\right)\right)}}_{\text{Sum of discounted rewards}}, \underbrace{\boxed{N(s,a)}}_{\text{state-action counter}} \right) \right]$$

  - Do not forget to update the models as well!

- Q-value Update (Model free)

Regular: $Q(s,a) \leftarrow Q(s,a) + \alpha \left(R(s,a,s') + \gamma \max_{a'}(Q(s',a')) - Q(s,a) \right)$

Modified: $Q(s,a) \leftarrow Q(s,a) + \alpha \left(R(s,a,s') + \gamma \max_{a'}(f(Q(s',a'), N(s',a')) - Q(s,a) \right)$

- These updates propagate the “bonus” back to states that lead to unknown states as well!
- At a given state, s, the agent performs the action that maximizes $V(s)$ or $Q(s,a)$

Some Other Ways to Pick the Action

- ε-greedy:
  - With some small chance (ε) pick a random action
  - Otherwise pick the action that gives the highest Q-value

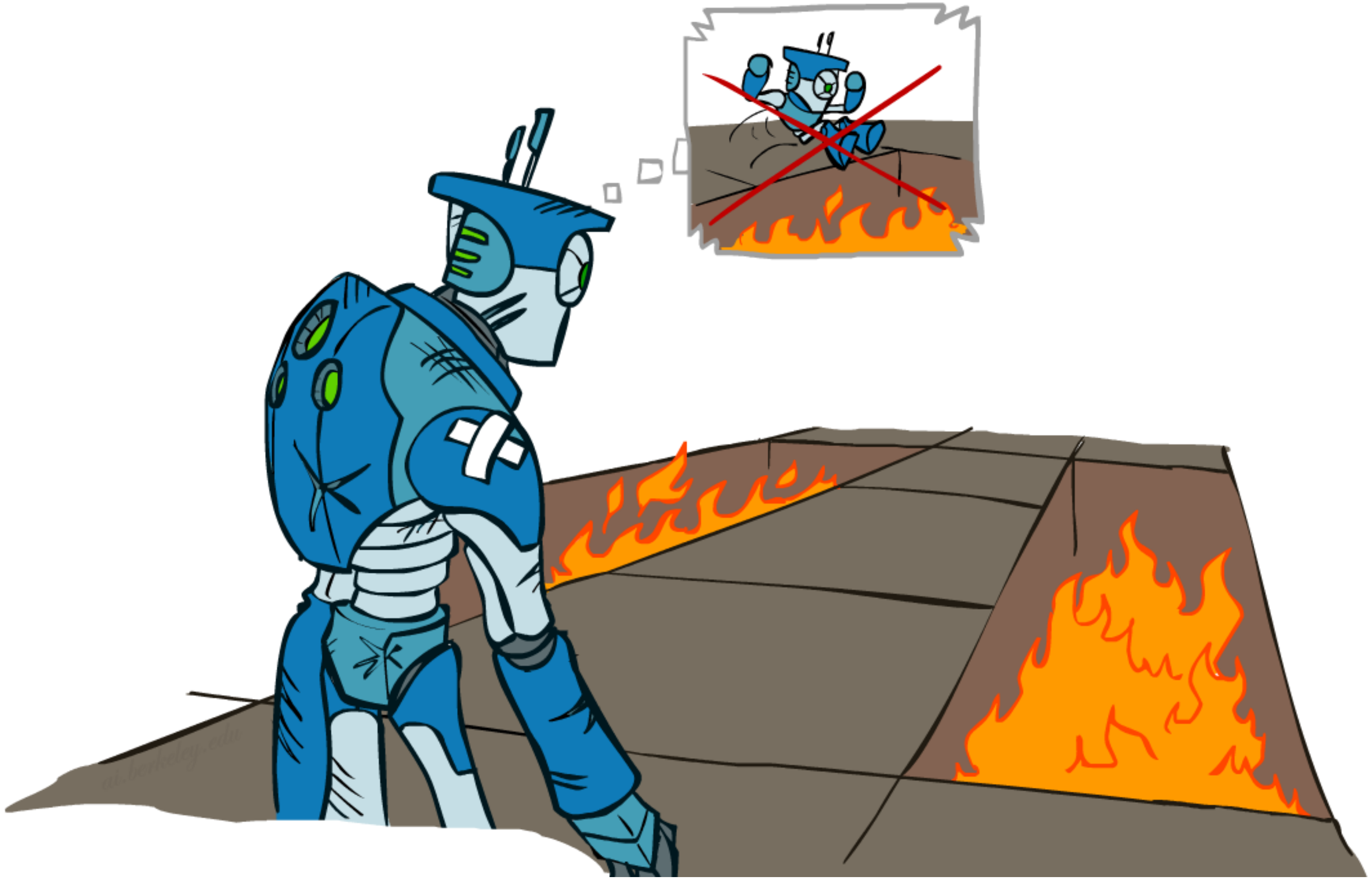
- Softmax: Pick the action with probability given by:

$$\mathrm{P}_{\pi}(\mathrm{a}|s) = \frac{e^{Q(s,a)}/\tau}{\sum_{a'} e^{Q(s,a')}/\tau}$$

  - This encourages the agent to pick more promising actions (high Q-value) more often
  - Need to eventually decrease τ
  - $\tau = \infty$: totally random, $\tau = 0$: totally greedy
  - Sample the actions from $\mathrm{P}_{\pi}(a|s)$
- Look up “Upper Confidence Bounds” for more sophisticated methods
- There are also other exploration function ideas

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



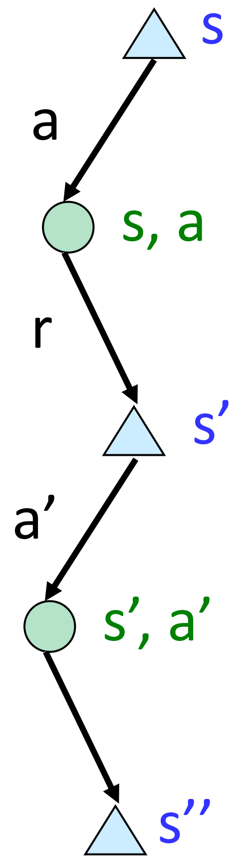
Minimizing regret is out of scope for this course but search for “Multi-Armed Bandits” if you are interested

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



Q-Learning

- We'd like to do Q-value updates to each Q-state:

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

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

- Instead, compute average as we go
  - Receive a sample transition (s,a,r,s')
  - This sample suggests

$$Q(s,a) \approx r + \gamma \max_{a'} Q(s',a')$$

  - But we want to average over results from (s,a) (Why?)
  - So keep a running average

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha) \left[r + \gamma \max_{a'} Q(s',a') \right]$$

Exploration: Back to ε-greedy

- Simplest: random actions (ε-greedy)
  - Every time step, flip a coin
  - With (small) probability ε, act randomly
  - With (large) probability 1-ε, act on current policy

- Yields the *soft* policy (given m actions)

$$\pi(a|s) = \begin{cases} 1 - \epsilon + \dfrac{\epsilon}{m} \text{if } \mathrm{a} = \underset{a}{\operatorname{argmax}}\left(\mathrm{Q}^{\pi_\mathrm{t}}(s, a)\right) \\ \dfrac{\epsilon}{m} \qquad \text{otherwise} \end{cases}$$

This is correct

Q-Learning with ε-greedy Action Selection

- Estimation/Target policy π is greedy w.r.t. $\mathrm{Q(S, A)}$

$$\pi(\mathrm{s}_{t+1}) = \underset{a'}{arg\mathrm{max}}(\mathrm{Q}(\mathrm{s}_{t+1}, a'))$$

- Chose the action via the ε-greedy policy with respect to $\mathrm{Q(S, A)}$

$$\pi(a|s) = \begin{cases} 1 - \epsilon + \dfrac{\epsilon}{m} \text{if } \mathrm{a} = \underset{a}{\mathrm{argmax}}\left(\mathrm{Q}^{\pi_{\mathrm{t}}}(s, a)\right) \\ \dfrac{\epsilon}{m} \qquad \text{otherwise} \end{cases}$$

- Use this action and observe the transition
- The target sample becomes:

$$sample = \mathrm{R_t} + \gamma \max_{a'}(\mathrm{Q}(\mathrm{s}_{t+1}, a'))$$

Q-Learning Pseudocode

```
function Q-Learning()
  //Q(s,a): Q-value estimates
  Qw() ← init() //randomly initialize all values except terminals
  forall episodes
    s ← s0 //get the start state, can change from episode to episode
    forall steps or until s is terminal
      a ← sampleTransition(s) //e.g. ε-greedy using Q
      r, s’ ← doAction(a) //observe a transition
      y ← r + γmaxa’(Q(s’, a’)) //Q(s’,a’)=0 if s’ is terminal
      Q(s,a) ← Q(s,a) + α(y-Q(s,a)) //or (1- α)Q(s,a) + αy
      s ← s’
```

Not shown: ε and α scheduling

Q-Learning Properties

- Amazing result: Q-learning converges to optimal policy -- even if you're acting suboptimally!

- This is called off-policy learning
- Caveats:
  - You have to explore enough
  - You have to eventually make the learning rate small enough
  - … but not decrease it too quickly
  - Basically, in the limit, it doesn't matter how you select actions (given exploration)
- Exploration – Exploitation Tradeoff

Model Based or Model Free

Unknown MDP: Model-Based

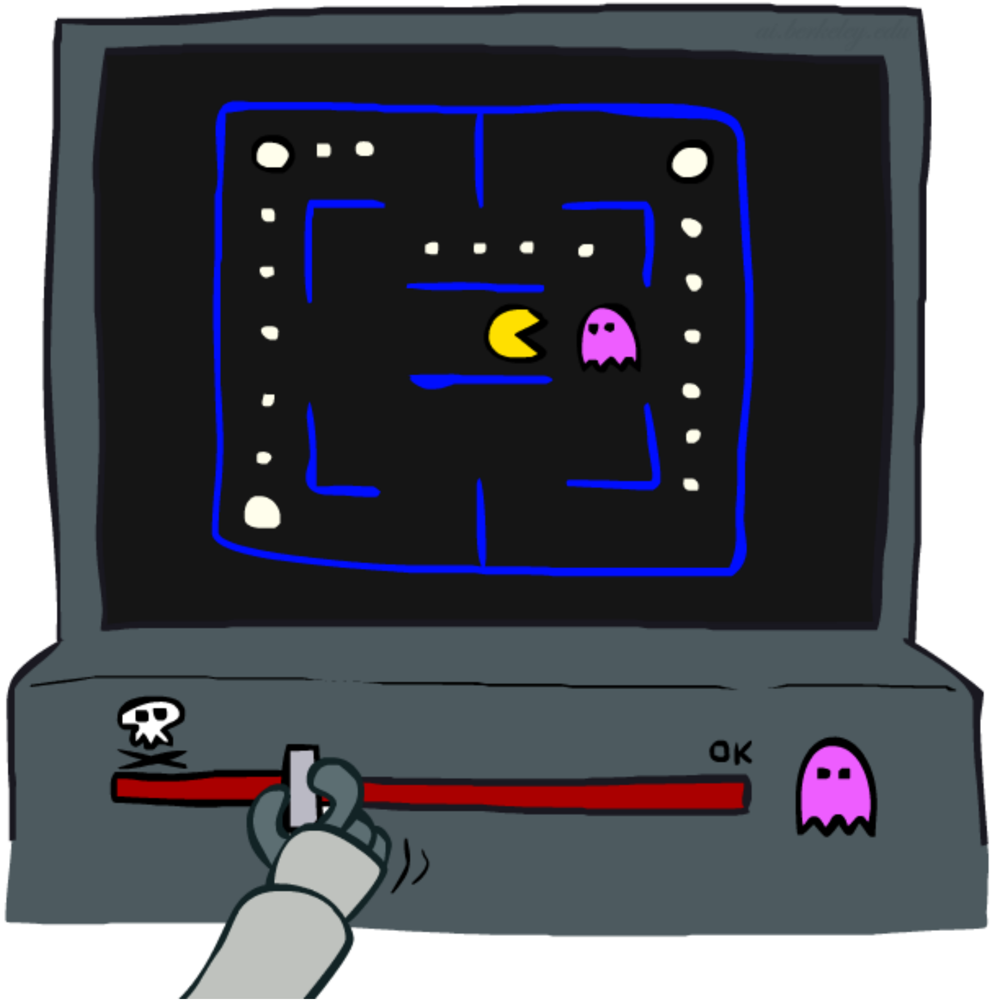
Goal	Technique
Compute V^*, Q^*, π^*	VI/PI on approx. MDP
Evaluate a fixed policy π	PE on approx. MDP

Unknown MDP: Model-Free

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

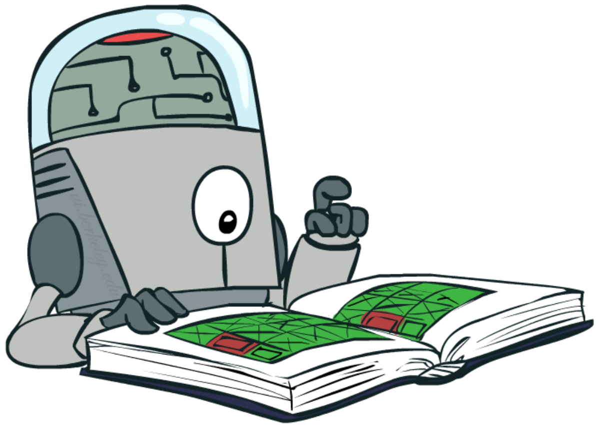
- In my opinion the answer is domain/problem dependent
- In practice, both approaches have solved interesting problems
- In some problems, model is already available or can be reliably estimated so a model-free approach does not make sense
- In some problems we do not want to re-calculate the policy every time we get a new percept, so a model-based approach does not make sense

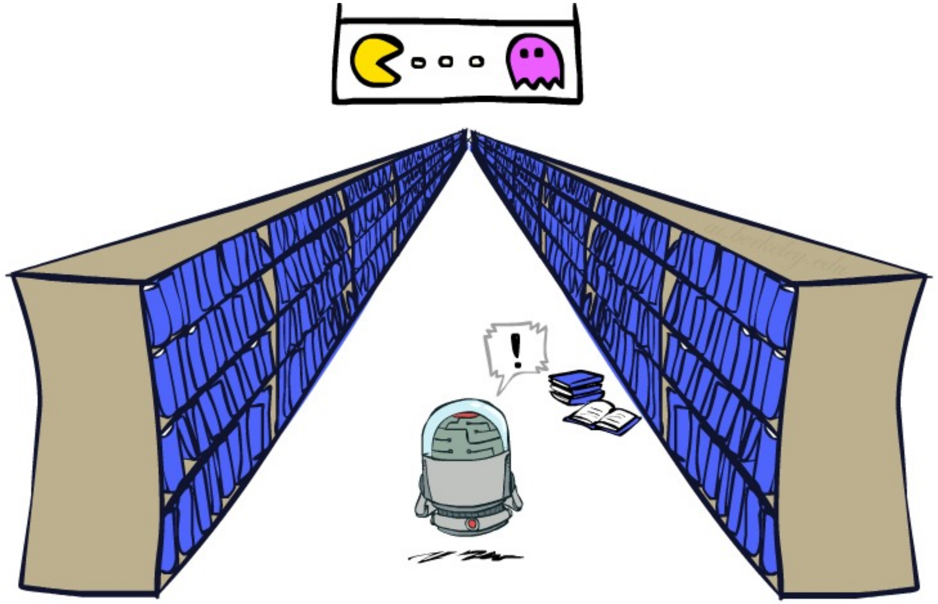
Approximate Q-Learning



Generalizing Across States

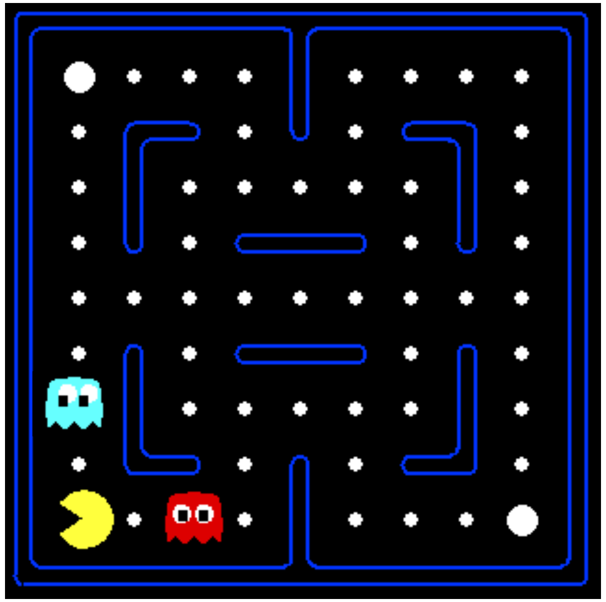
- Basic Q-Learning keeps a table of all q-values
- In realistic situations, we cannot possibly learn about every single state!
  - Too many states to visit them all in training
  - Too many states to hold the q-tables in memory
- Instead, we want to generalize:
  - Learn in a smaller space from experience
  - Generalize that experience to new, similar situations
  - This is the same as the fundamental idea in machine learning



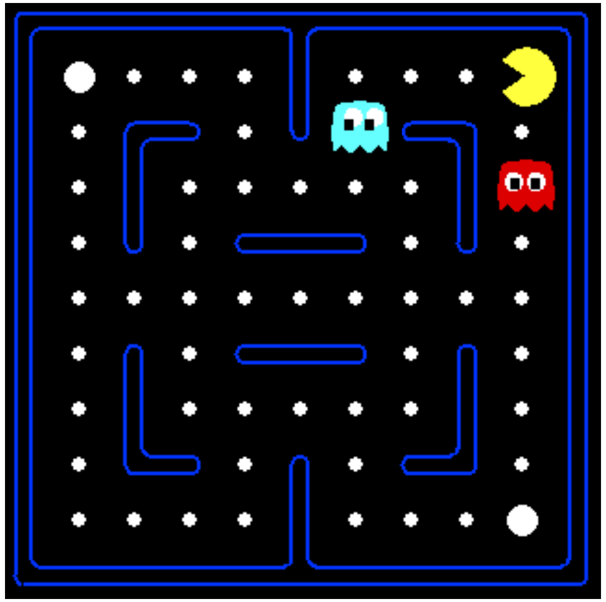


Example: Pacman

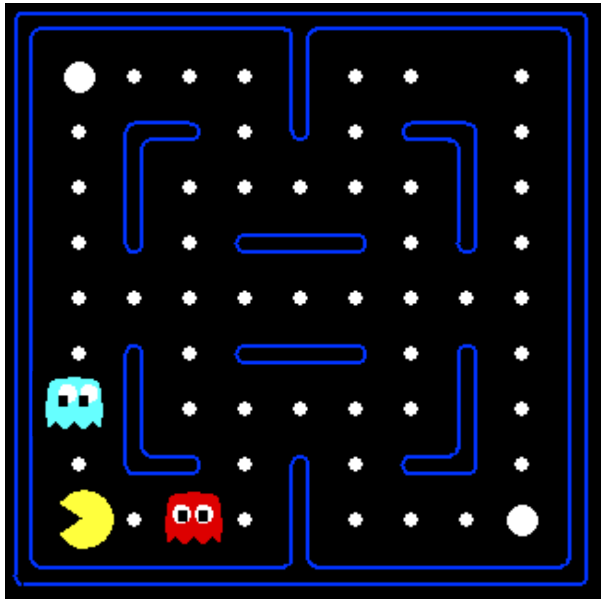
Let's say we discover through experience that this state is bad:



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

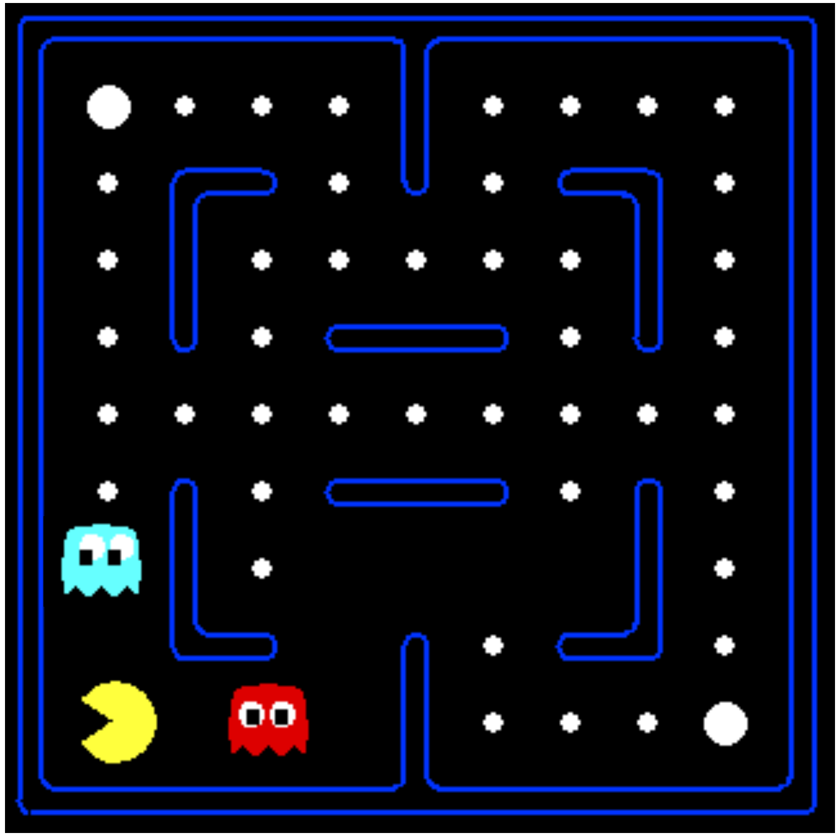


Or even this one!



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:

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

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

- 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

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

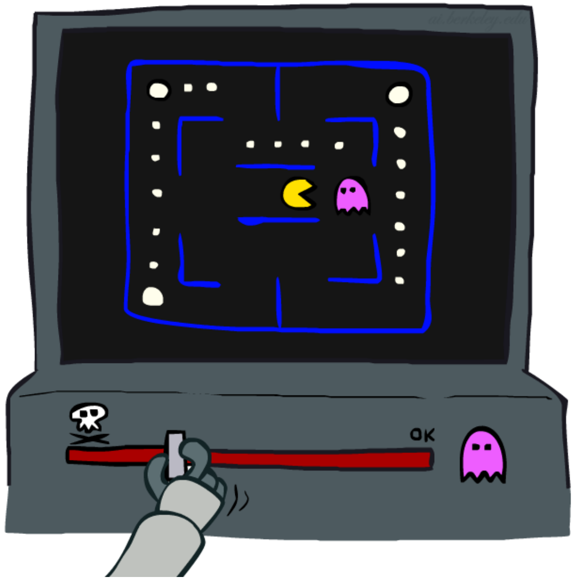
- Q-learning with linear Q-functions:

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

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

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

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

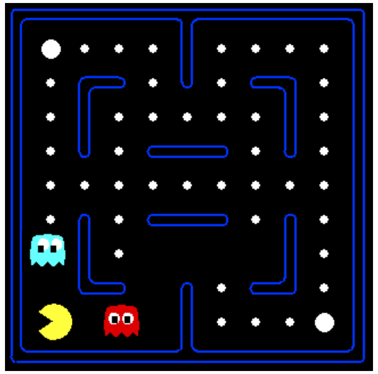


- Intuitive interpretation:
  - Adjust weights of active features
  - E.g., if something unexpectedly bad happens, blame the features that were ON: disprefer all states with that state's features
- Formal justification: online least squares

Example: Q-Pacman

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

s

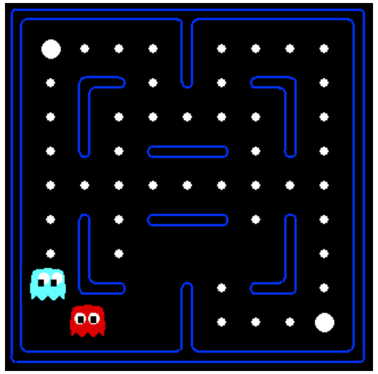


$f_{DOT}(s, \text{NORTH}) = 0.5$

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

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

s'



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

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

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

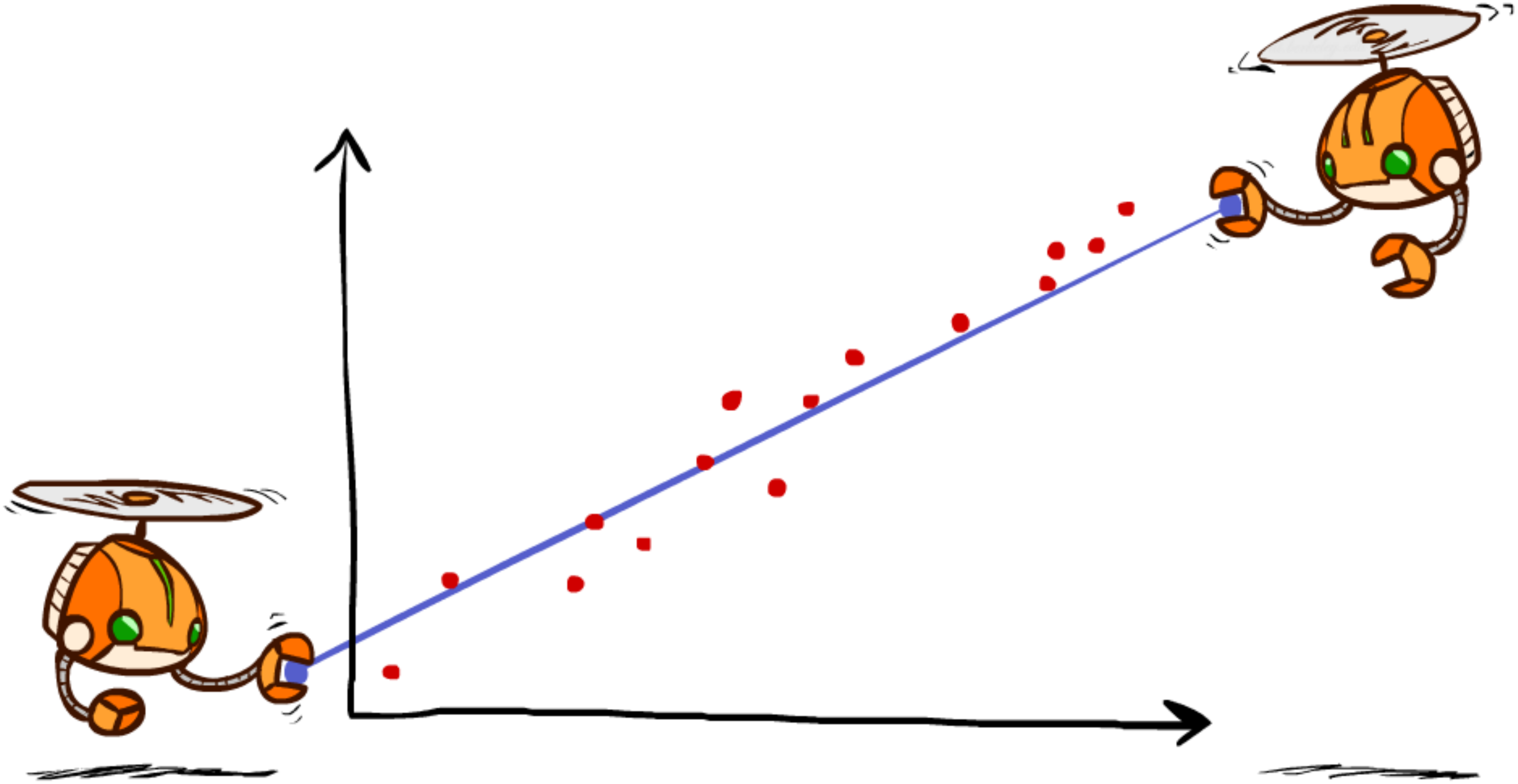
difference $= -501$

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

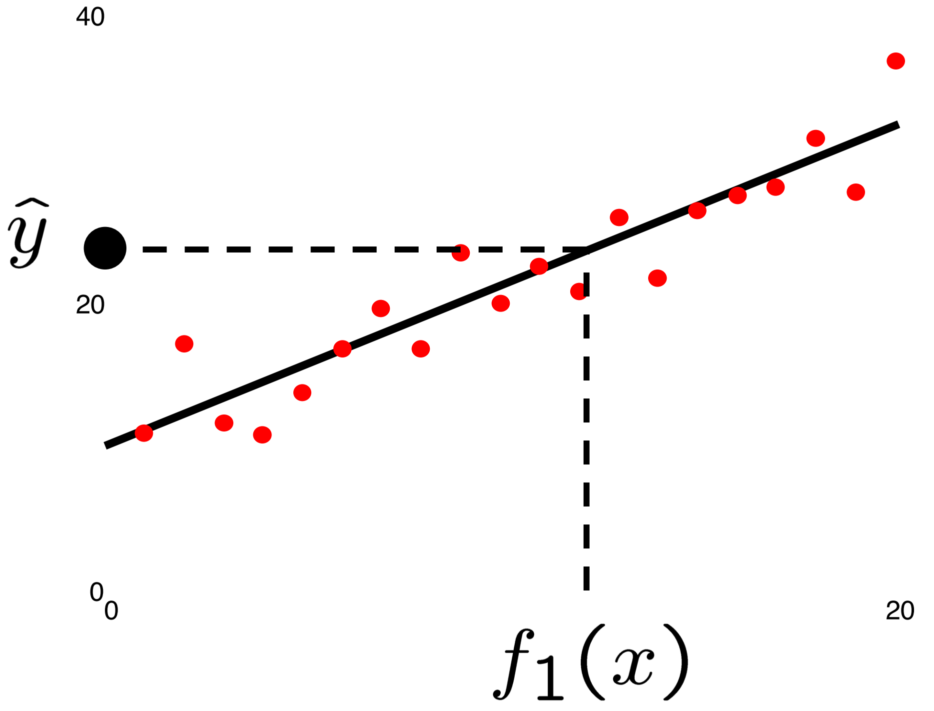
$w_{GST} \leftarrow -1.0 + \alpha\,[-501]\,1.0$

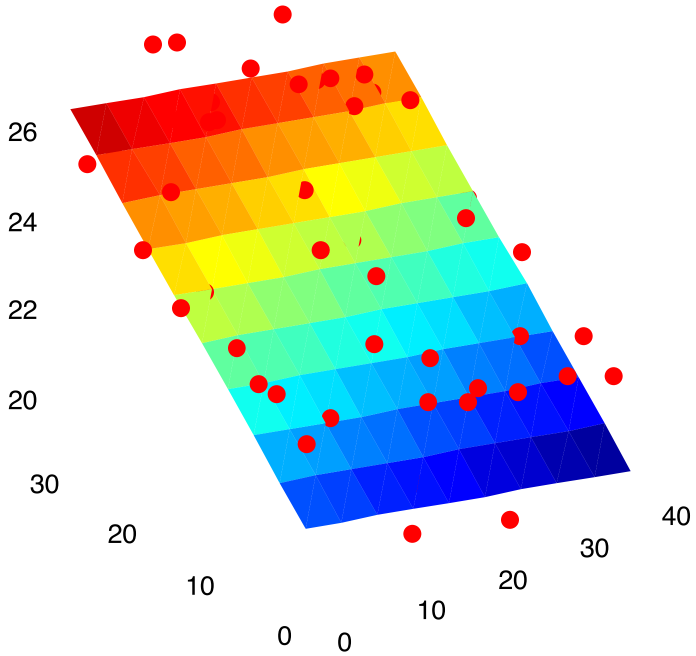
$$Q(s,a) = 3.0 f_{DOT}(s,a) - 3.0 f_{GST}(s,a)$$

Q-Learning and Least Squares



Linear Approximation: Regression*





Prediction:

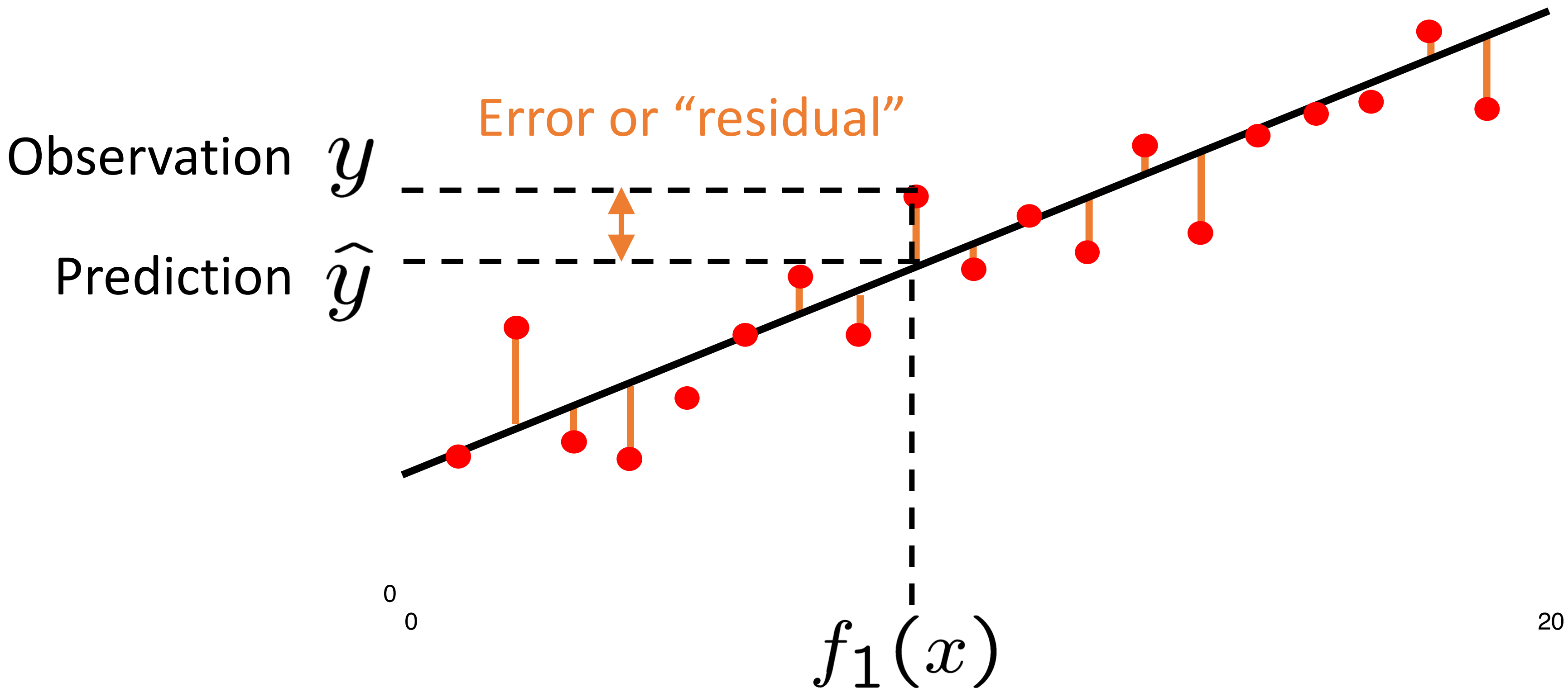
$$\hat{y} = w_0 + w_1 f_1(x)$$

Prediction:

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

Optimization: Least Squares*

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



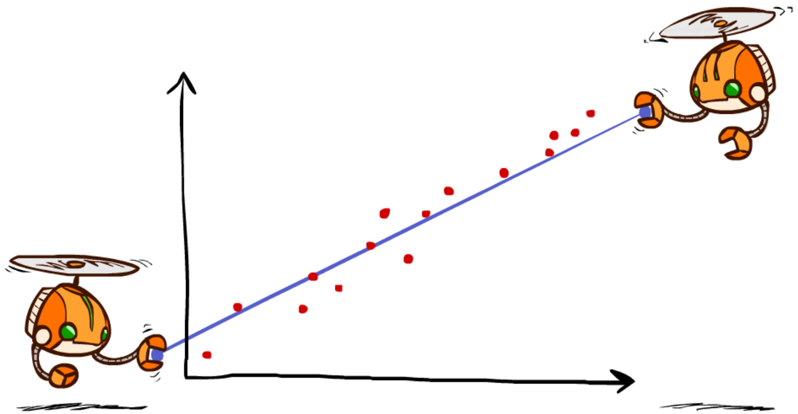
Minimizing Error*

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"

Q-Learning with Linear Function Approximation

```
function Q-Learning_LinFunApprox()
  //Qw(s,a): approximate Q-value = wTf(s,a) = w1f1(s,a) + … + wnfn(s,a)
  w ← init() //random
  forall episodes
    s ← s0 //get the start state, can change from episode to episode
    forall steps or until s is terminal
      a ← sampleTransition(s) //e.g. ε-greedy using Qw
      r, s’ ← doAction(a) //observe a transition
      y ← r + γmaxa’(Qw(s’, a’)) //Qw(s’,a’)=0 if s’ is terminal
      w ← w + αw(y-Qw(s,a))f( s,a) //this is a vector operation
      s ← s’
```

Not shown: ε and α_w scheduling

Policy Search

- Problem: often the feature-based policies that work well (win games, maximize utilities) aren't the ones that approximate V / Q best
  - E.g. your value functions from project 2 were probably horrible estimates of future rewards, but they still produced good decisions
  - Q-learning's priority: get Q-values close (modeling)
  - Action selection priority: get ordering of Q-values right (prediction)

- 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

Reinforcement Learning

- Exact RL:
  - Too big to represent and to learn
  - No generalization between similar states (bigger issue!)
- Approximate RL (aka value function approximation):
  - Extract features from the state (compression)
  - Represent V or Q with a function approximator (mostly parametric, usually linear or NN-based)
  - Allows for **generalization** between states if "<u>features are good</u>"
- Policy Search
  - At the end of the day, we mostly care about the policy not the values
  - Learning parameters of a policy representation

Policy Search

[Andrew Ng]