

Lecture 7: Reinforcement Learning

Shuai Li

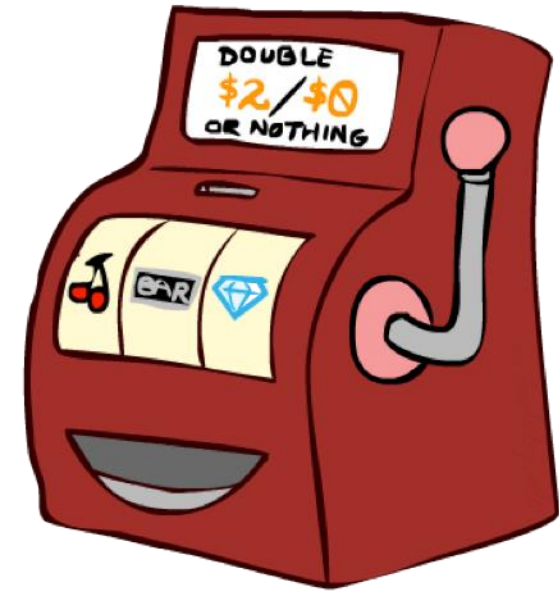
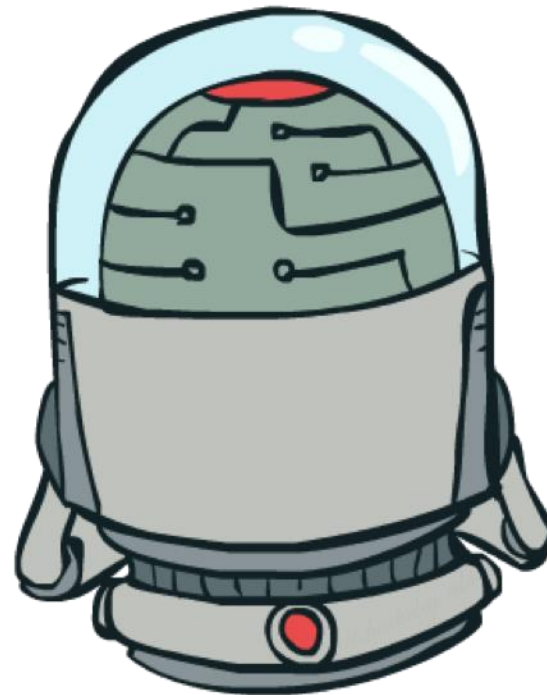
John Hopcroft Center, Shanghai Jiao Tong University

<https://shuaili8.github.io>

<https://shuaili8.github.io/Teaching/CS410/index.html>

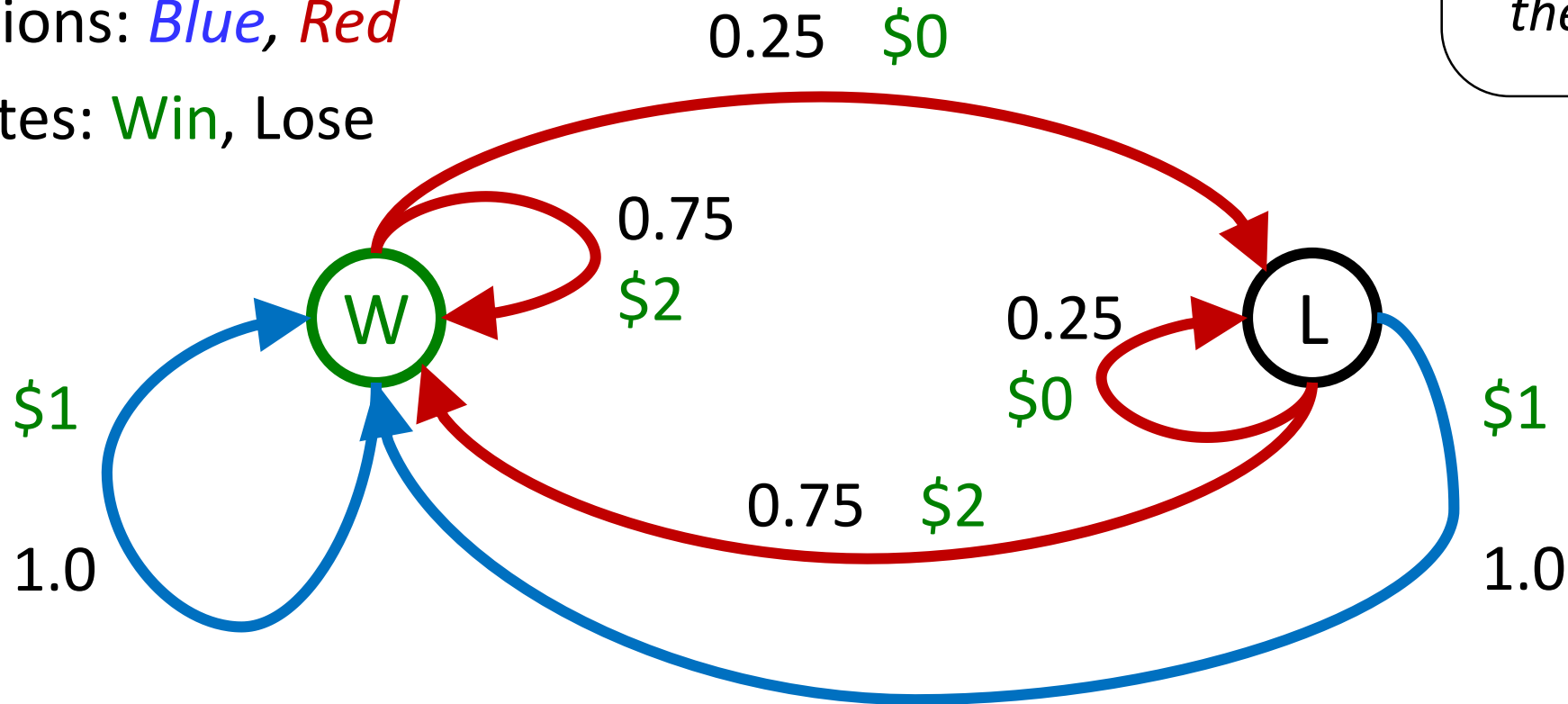
Part of slide credits: CMU AI & <http://ai.berkeley.edu>

Example: Double Bandits



Example: Double Bandits - MDP

- Actions: *Blue, Red*
- States: *Win, Lose*



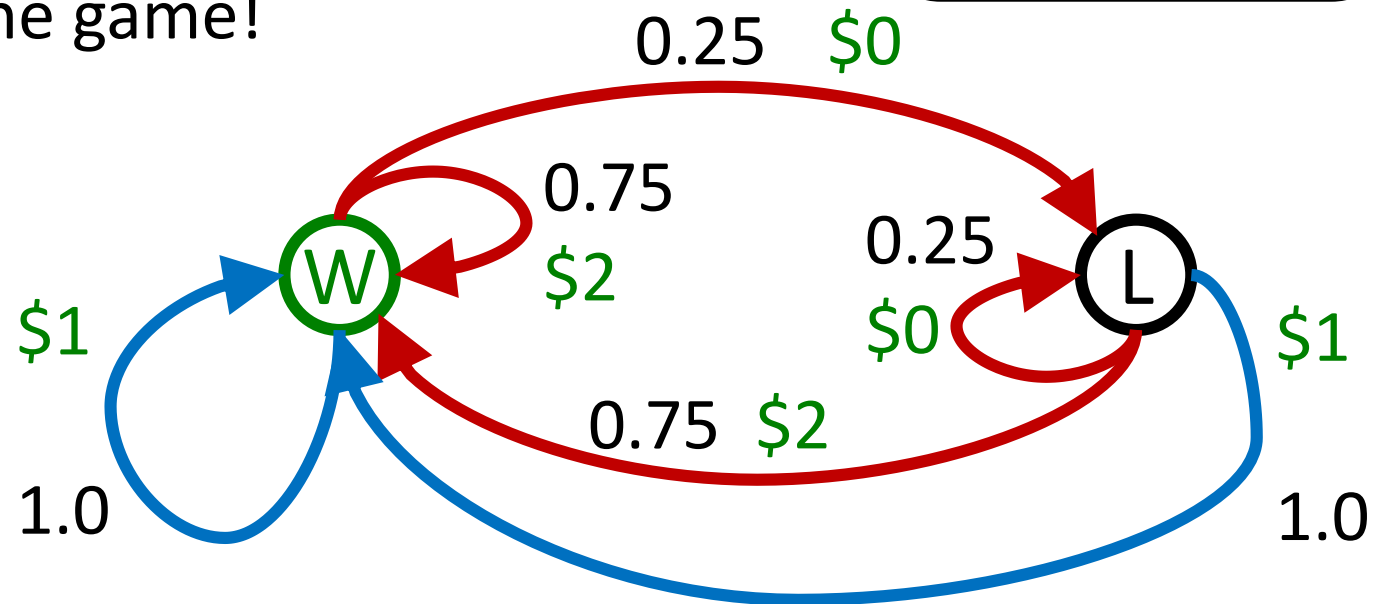
No discount
100 time steps
Both states have the same value

Example: Double Bandits - Offline Planning

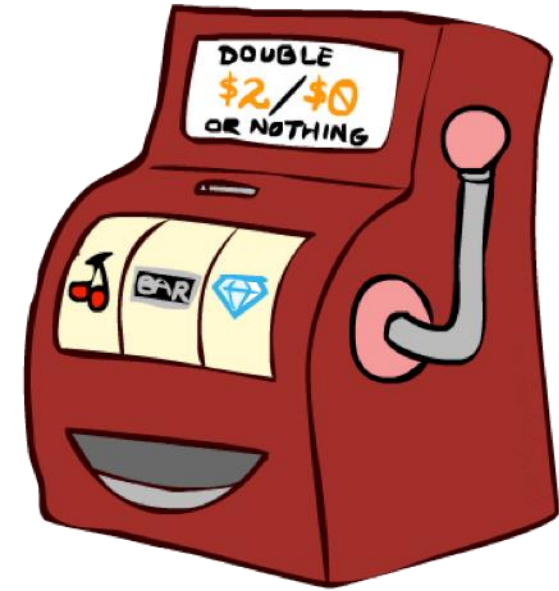
- Solving MDPs is offline planning
 - You determine all quantities through computation
 - You need to know the details of the MDP
 - You do not actually play the game!

No discount
100 time steps
Both states have the same value

	Value
Play Red	150
Play Blue	100



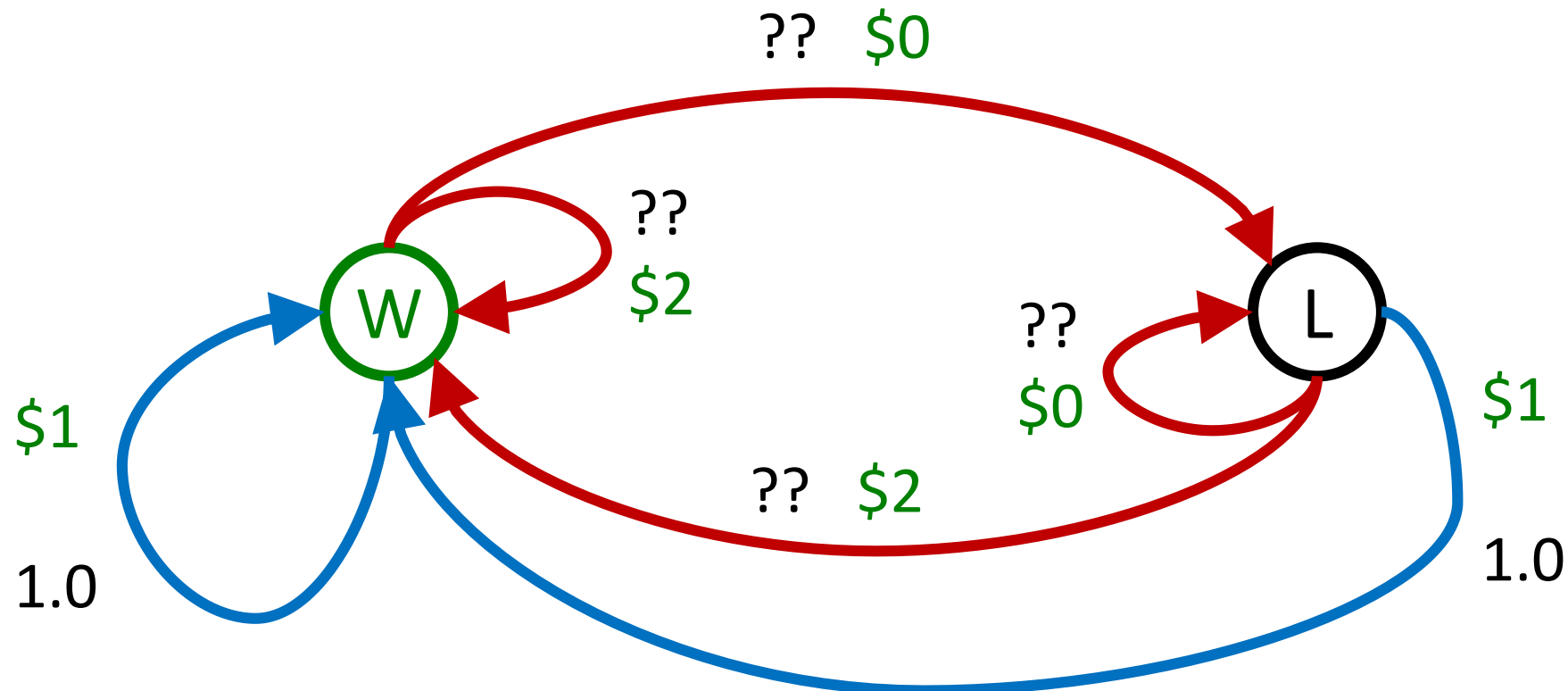
Example: Double Bandits - Let's Play!



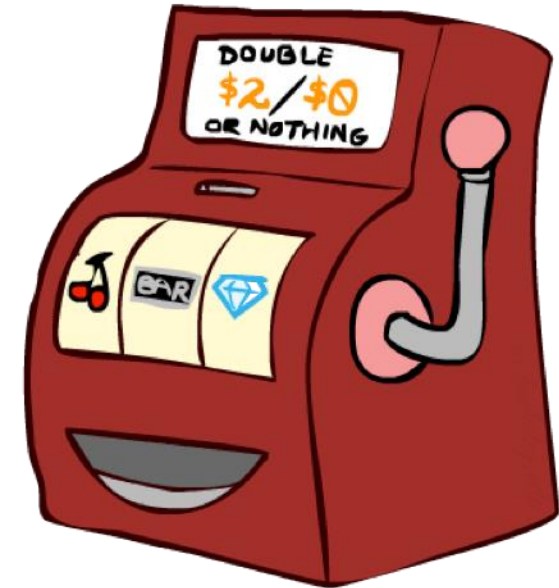
\$2 \$2 \$0 \$2 \$2
\$2 \$2 \$0 \$0 \$0

Example: Double Bandits - Online Planning

- Rules changed! Red's win chance is different.



Example: Double Bandits - Let's Play!



\$0 \$0 \$0 \$2 \$0
\$2 \$0 \$0 \$0 \$0

What Just Happened?



- That wasn't planning, it was learning!
 - Specifically, reinforcement learning
 - There was an MDP, but you couldn't solve it with just computation
 - You needed 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**: even if you learn intelligently, you make mistakes
 - **Sampling**: because of chance, you have to try things repeatedly
 - **Difficulty**: learning can be much harder than solving a known MDP

Reinforcement Learning

- What if we didn't know $P(s'|s, a)$ and $R(s, a, s')$?

Value iteration:
$$V_{k+1}(s) = \max_a \sum_{s'} P(s'|s, a) [R(s, a, s') + \gamma V_k(s')], \quad \forall s$$

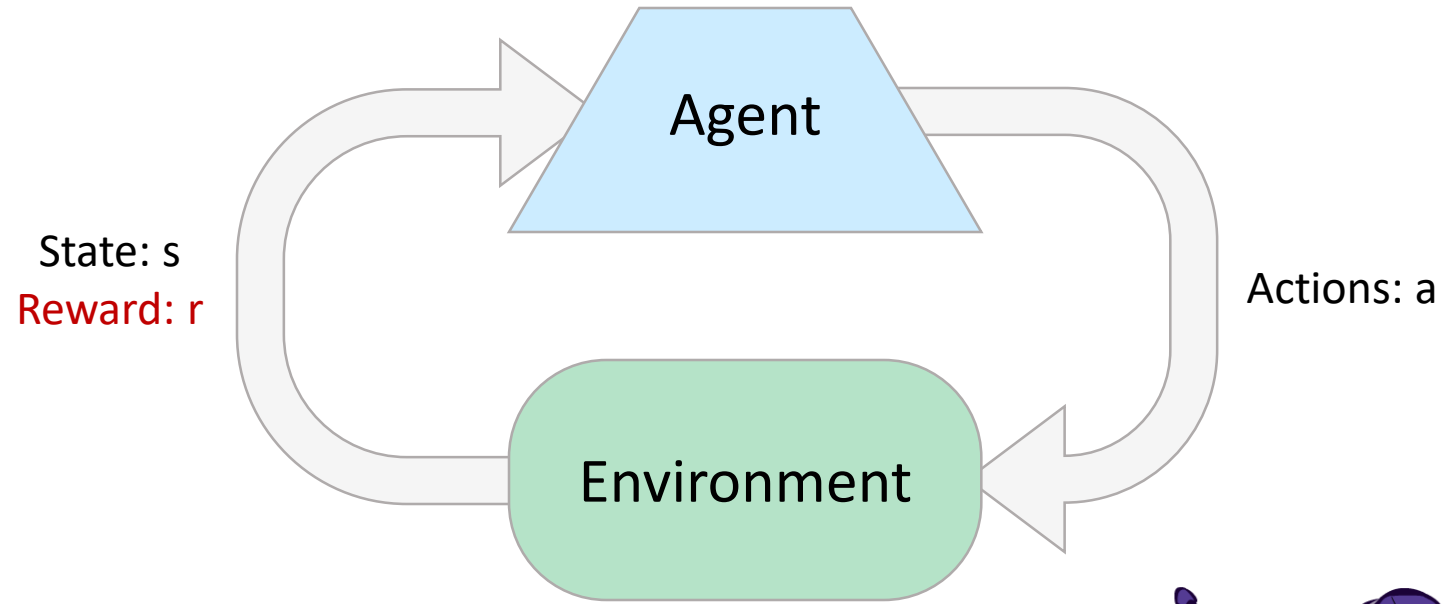
Q-iteration:
$$Q_{k+1}(s, a) = \sum_{s'} P(s'|s, a) [R(s, a, s') + \gamma \max_{a'} Q_k(s', a')], \quad \forall s, a$$

Policy extraction:
$$\pi_V(s) = \operatorname{argmax}_a \sum_{s'} P(s'|s, a) [R(s, a, s') + \gamma V(s')], \quad \forall s$$

Policy evaluation:
$$V_{k+1}^\pi(s) = \sum_{s'} P(s'|s, \pi(s)) [R(s, \pi(s), s') + \gamma V_k^\pi(s')], \quad \forall s$$

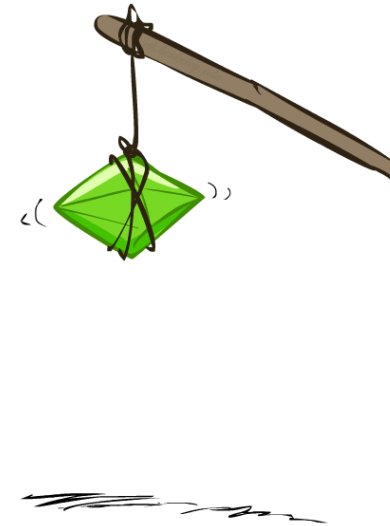
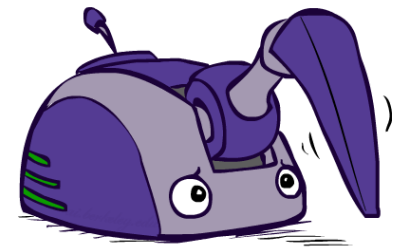
Policy improvement:
$$\pi_{new}(s) = \operatorname{argmax}_a \sum_{s'} P(s'|s, a) [R(s, a, s') + \gamma V^{\pi_{old}}(s')], \quad \forall s$$

Reinforcement Learning 2



- 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!



Reinforcement Learning 3

- Still assume a Markov decision process (MDP):

- A set of states $s \in S$
- A set of actions (per state) A
- A model $T(s,a,s')$
- A reward function $R(s,a,s')$

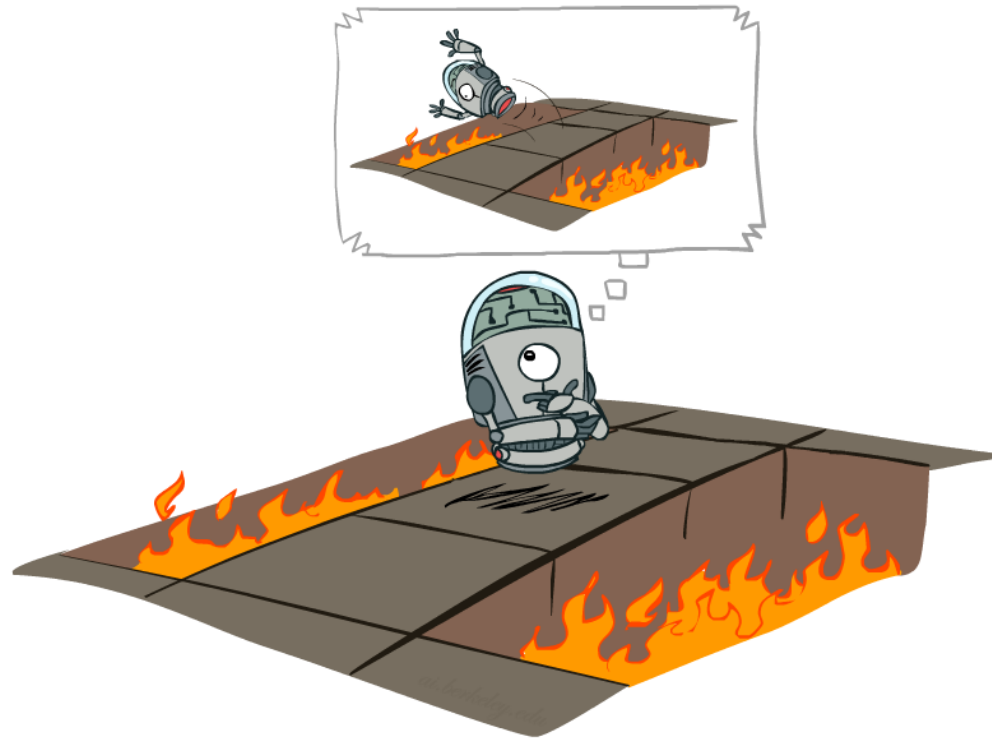
- Still looking for a policy $\pi(s)$

- 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



Offline (MDPs) vs. Online (RL)



Offline Solution



Online Learning

Example: Learning to Walk



Initial



A Learning Trial



After Learning [1K Trials]

Example: Learning to Walk 2



Initial

Example: Learning to Walk 3



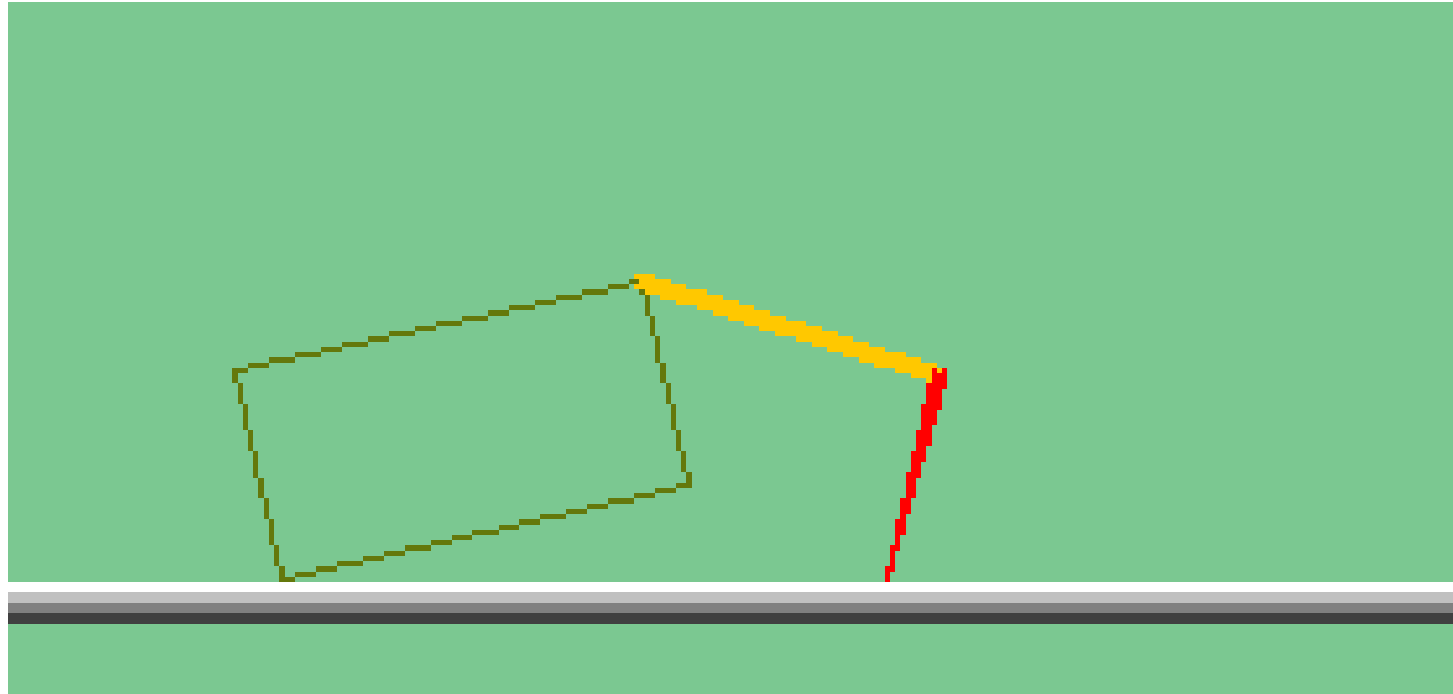
Training

Example: Learning to Walk 4

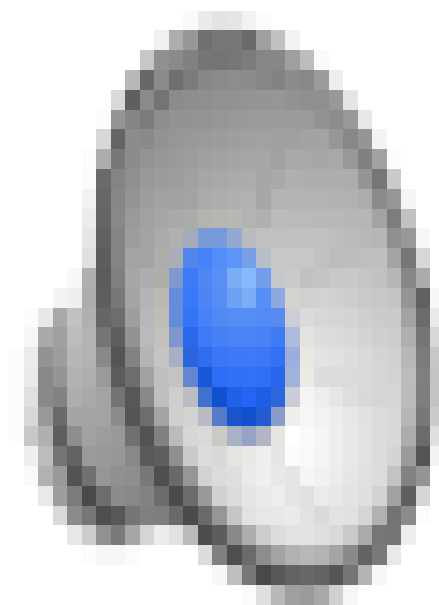


Finished

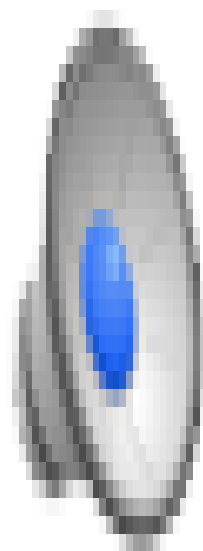
Example: The Crawler!



Video of Demo Crawler Bot

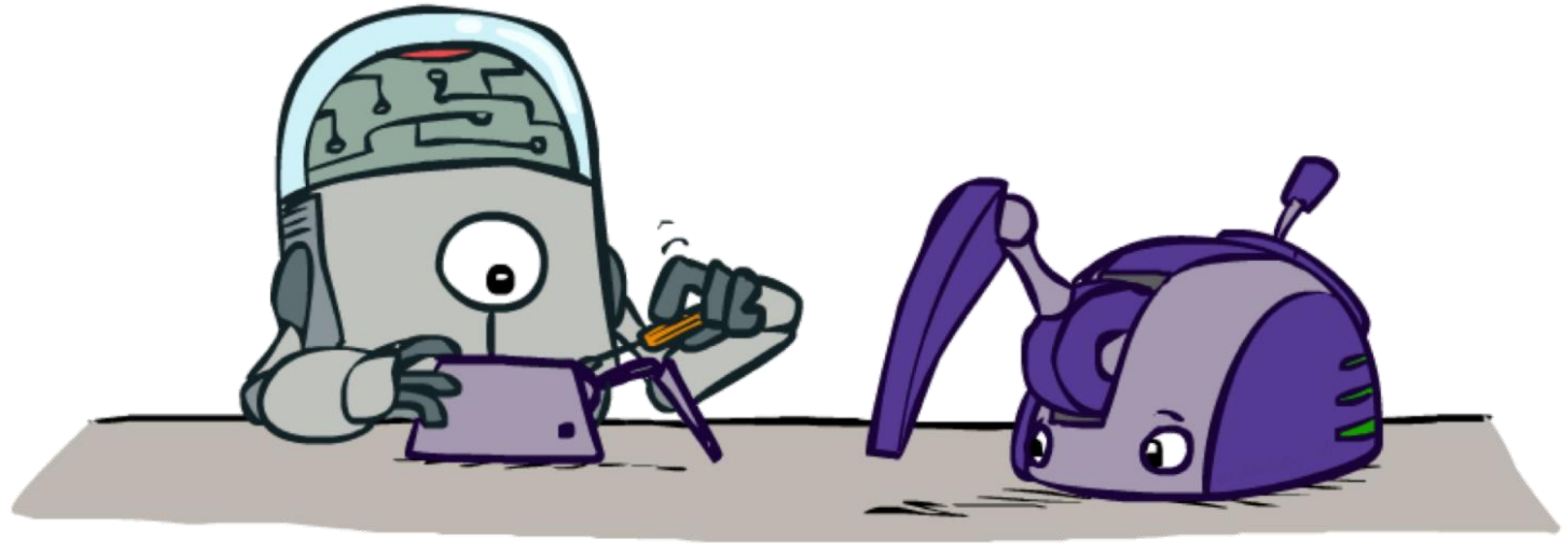


DeepMind Atari (©Two Minute Lectures)



Reinforcement Learning -- Overview

- **Passive Reinforcement Learning (= how to learn from experiences)**
 - **Model-based Passive RL**
 - Learn the MDP model from experiences, then solve the MDP
 - **Model-free Passive RL**
 - Forego learning the MDP model, directly learn V or Q:
 - Value learning – learns value of a fixed policy; 2 approaches: Direct Evaluation & TD Learning
 - Q learning – learns Q values of the optimal policy (uses a Q version of TD Learning)
- **Active Reinforcement Learning (= agent also needs to decide how to collect experiences)**
 - **Key challenges:**
 - How to efficiently explore?
 - How to trade off exploration <> exploitation
 - Applies to both model-based and model-free.
we'll cover only in context of Q-learning



Model-Based Learning

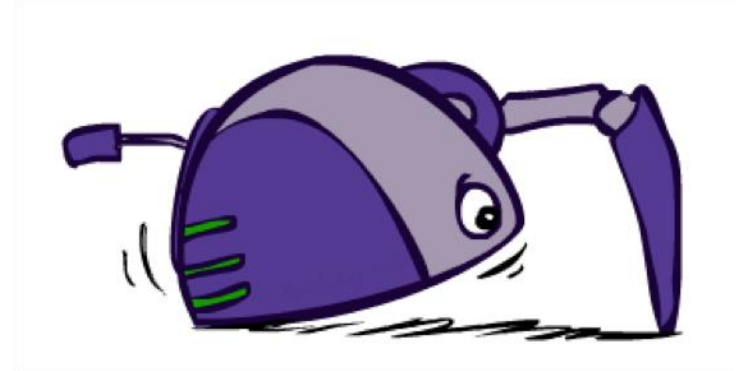
Reinforcement Learning -- Overview

- **Passive Reinforcement Learning (= how to learn from experiences)**
 - **Model-based Passive RL**
 - Learn the MDP model from experiences, then solve the MDP
 - **Model-free Passive RL**
 - Forego learning the MDP model, directly learn V or Q:
 - Value learning – learns value of a fixed policy; 2 approaches: Direct Evaluation & TD Learning
 - Q learning – learns Q values of the optimal policy (uses a Q version of TD Learning)
- **Active Reinforcement Learning (= agent also needs to decide how to collect experiences)**
 - **Key challenges:**
 - How to efficiently explore?
 - How to trade off exploration <> exploitation
 - Applies to both model-based and model-free.
we'll cover only in context of Q-learning

Model-Based Reinforcement Learning

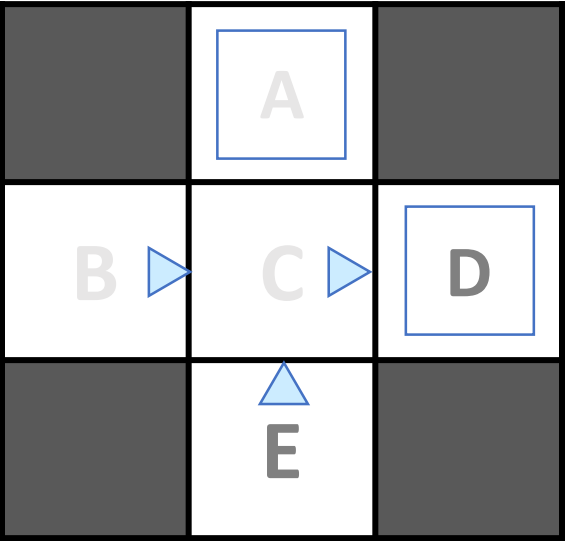
- Model-Based Idea:
 - Learn an approximate model based on experiences
 - Solve for values as if the learned model were correct
- Step 1: Learn empirical MDP model
 - Count outcomes s' for each s, a
 - Normalize to give an estimate of $\hat{T}(s, a, s')$
 - Discover each $\hat{R}(s, a, s')$ when we experience (s, a, s')
- Step 2: Solve the learned MDP
 - For example, use value iteration, as before

(and repeat as needed)



Example: Model-Based RL

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
 ...

Analogy: Expected Age

Goal: Compute expected age of students

Known $P(A)$

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

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

Unknown $P(A)$: "Model Based"

$$\hat{P}(a) = \frac{\text{num}(a)}{N}$$
$$E[A] \approx \sum_a \hat{P}(a) \cdot a$$

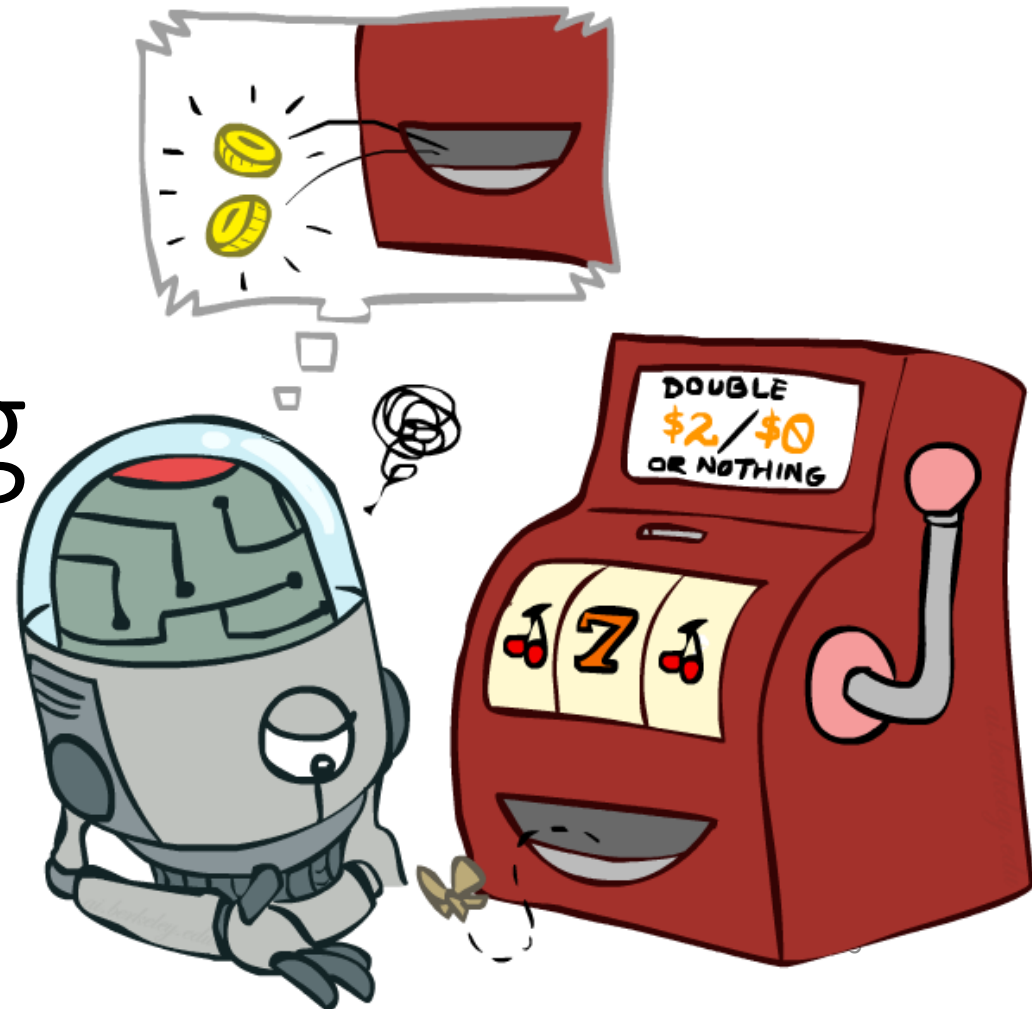
Why does this work? Because eventually you learn the right model.

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



Reinforcement Learning -- Overview

- Passive Reinforcement Learning (= how to learn from experiences)
 - Model-based Passive RL
 - Learn the MDP model from experiences, then solve the MDP
 - Model-free Passive RL
 - Forego learning the MDP model, directly learn V or Q:
 - Value learning – learns value of a fixed policy; 2 approaches: Direct Evaluation & TD Learning
 - Q learning – learns Q values of the optimal policy (uses a Q version of TD Learning)
- Active Reinforcement Learning (= agent also needs to decide how to collect experiences)
 - Key challenges:
 - How to efficiently explore?
 - How to trade off exploration <> exploitation
 - Applies to both model-based and model-free.
we'll cover only in context of Q-learning

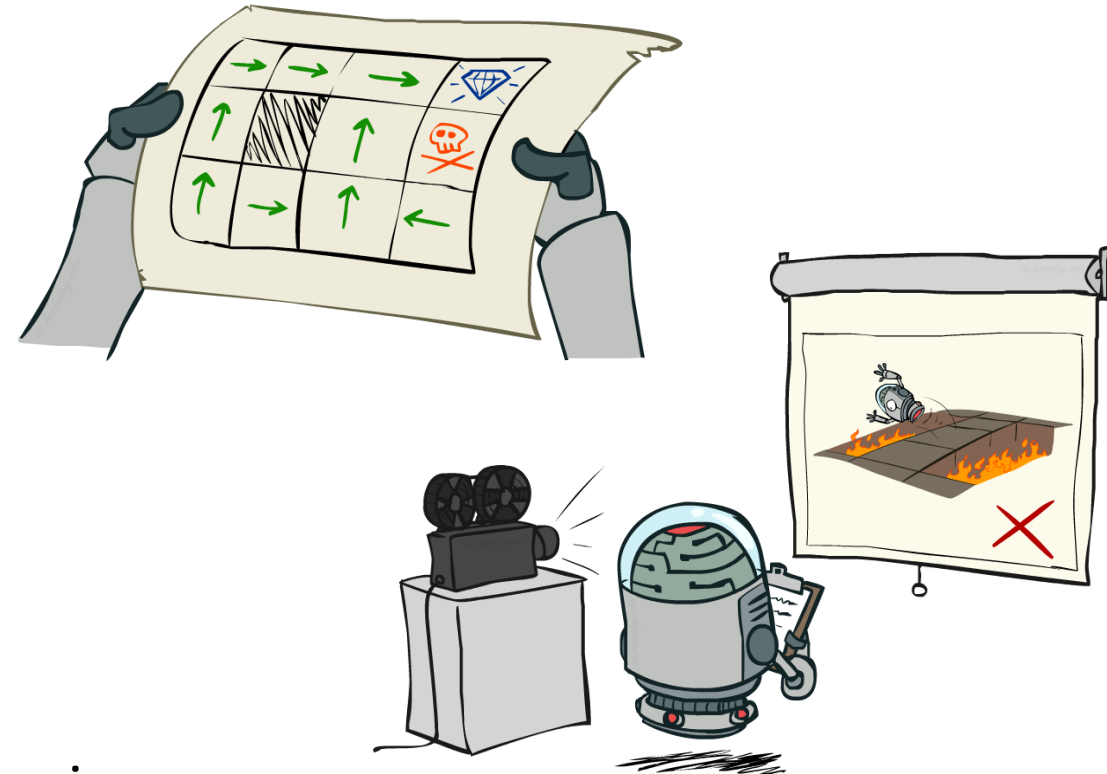
Passive Model-Free Reinforcement Learning

- 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**

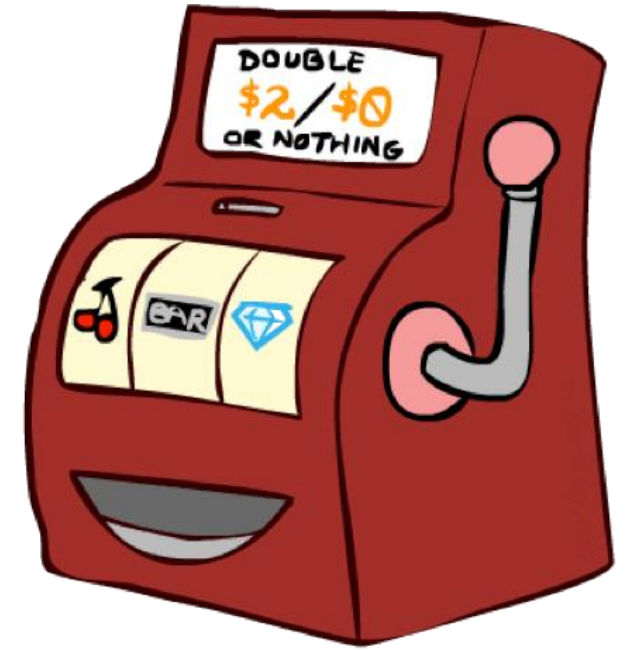
- In this case:

- 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



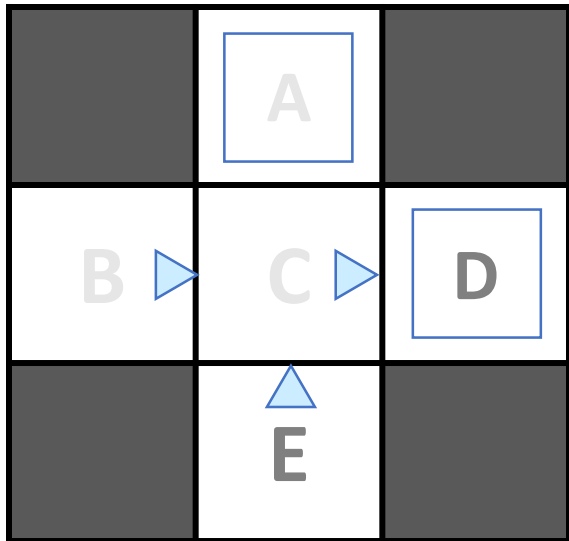
Direct Evaluation

- Goal: Compute values for each state under π
- 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



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

	-10	
	A	
+8	+4	+10
B	C	D
	-2	
	E	

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

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 wastes information about state connections
 - Each state must be learned separately
 - So, it takes a long time to learn

Output Values

	-10 A	
+8 B	+4 C	+10 D
	-2 E	

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

Reinforcement Learning -- Overview

- Passive Reinforcement Learning (= how to learn from experiences)
 - Model-based Passive RL
 - Learn the MDP model from experiences, then solve the MDP
 - Model-free Passive RL
 - Forego learning the MDP model, directly learn V or Q:
 - Value learning – learns value of a fixed policy; 2 approaches: Direct Evaluation & [TD Learning](#)
 - Q learning – learns Q values of the optimal policy (uses a Q version of TD Learning)
- Active Reinforcement Learning (= agent also needs to decide how to collect experiences)
 - Key challenges:
 - How to efficiently explore?
 - How to trade off exploration <> exploitation
 - Applies to both model-based and model-free.
we'll cover only in context of Q-learning

Why Not Use Policy Evaluation?

- Simplified Bellman updates calculate V for a fixed policy:

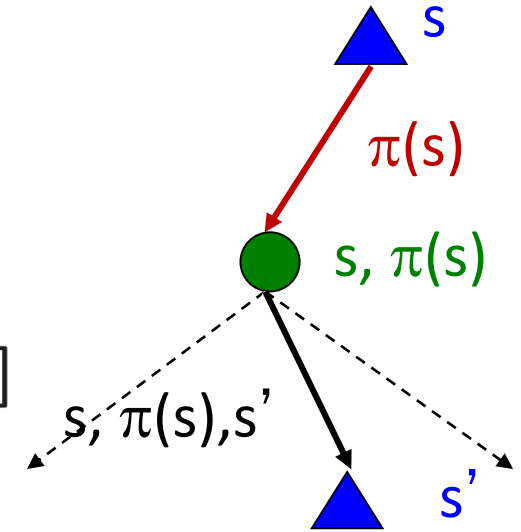
- Each round, replace V with a one-step-look-ahead layer over V

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

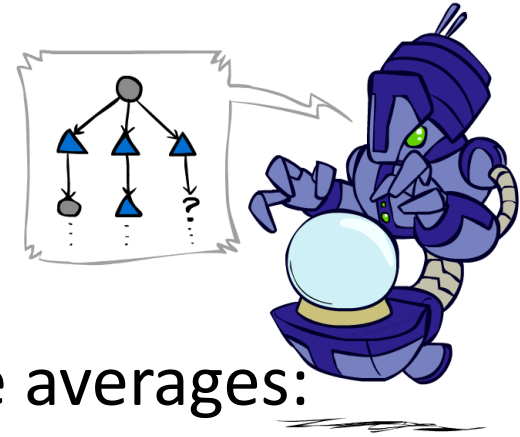
$$V_{k+1}^\pi(s) \leftarrow \sum_{s'} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V_k^\pi(s')]$$

- 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 do 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:

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

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

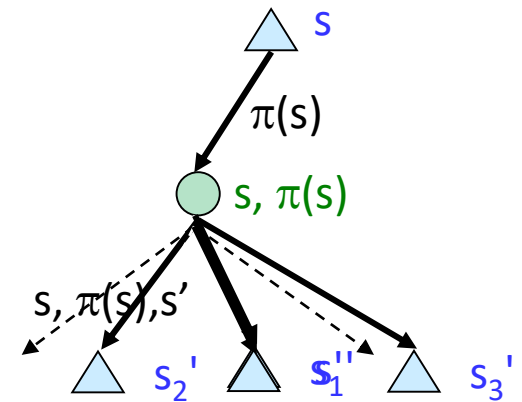
$$\text{sample}_1 = R(s, \pi(s), s'_1) + \gamma V_k^{\pi}(s'_1)$$

$$\text{sample}_2 = R(s, \pi(s), s'_2) + \gamma V_k^{\pi}(s'_2)$$

...

$$\text{sample}_n = R(s, \pi(s), s'_n) + \gamma V_k^{\pi}(s'_n)$$

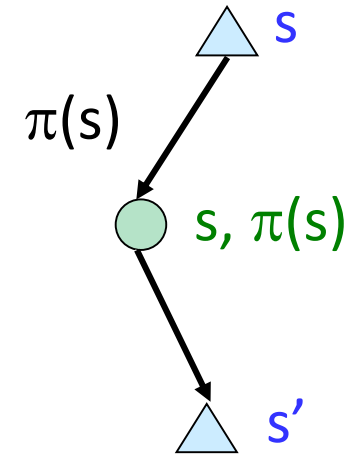
$$V_{k+1}^{\pi}(s) \leftarrow \frac{1}{n} \sum_i \text{sample}_i$$



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

Temporal Difference Value 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: $V^\pi(s) \leftarrow V^\pi(s) + \alpha(sample - V^\pi(s))$

Gradient Descent View

$$f(x) = \frac{1}{2}(y - x)^2$$

$$\frac{df}{dx} = -(y - x)$$

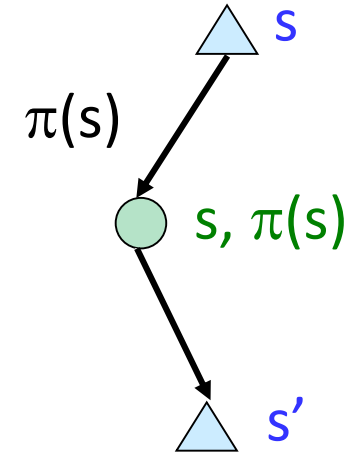
- Goal: find x that minimizes $f(x)$
 1. Start with initial guess, x_0
 2. Update x by taking a step in the direction that $f(x)$ is changing fastest (in the negative direction) with respect to x :
 $x \leftarrow x - \alpha \nabla_x f$, where α is the step size or learning rate
 3. Repeat until convergence
- TD goal: find value(s), V , that minimizes difference between sample(s) and V

$$V \leftarrow V - \alpha \nabla_V Error$$

$$Error(V) = \frac{1}{2} (sample - V)^2$$

Gradient Descent View 2

- 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 $V(s)$: $sample = r + \gamma V^\pi(s')$

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

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

Same update: $V^\pi(s) \leftarrow V^\pi(s) - \alpha \nabla Error$ $Error = \frac{1}{2} (sample - V_{37}^\pi(s))^2$

Exponential Moving Average

- Exponential moving average

- The running interpolation update: $V_n = (1 - \alpha)V_{n-1} + \alpha x_n$ with $V_1 = x_1$

- Makes recent samples more important

$$V_n = \alpha x_n + \alpha(1 - \alpha)x_{n-1} + \dots + \alpha(1 - \alpha)^{n-2}x_2 + (1 - \alpha)^{n-1}x_1$$

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

- Decreasing learning rate (alpha) can give converging averages

- Note $V_n = \alpha_n x_n + (1 - \alpha_n)\alpha_{n-1}x_{n-1} + \dots$
 $+ (1 - \alpha_n)(1 - \alpha_{n-1}) \cdot \dots \cdot (1 - \alpha_3)\alpha_2 x_2$
 $+ (1 - \alpha_n)(1 - \alpha_{n-1}) \cdot \dots \cdot (1 - \alpha_3)(1 - \alpha_2)x_1$

Example: Temporal Difference Value Learning

States

	A	
B	C	D
	E	

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

Observed Transitions

B, east, C, -2

	0	
0	0	8
	0	

C, east, D, -2

	0	
-1	0	8
	0	

	0	
-1	3	8
	0	

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

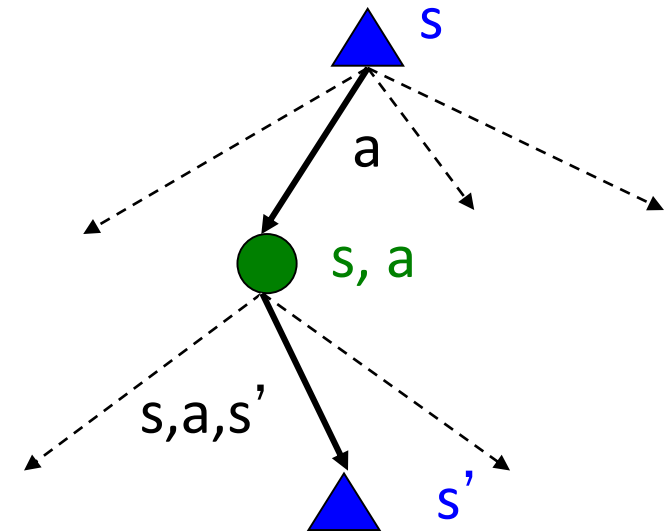
Problems with TD Value Learning

- TD value learning 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:

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

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

- Idea: learn Q-values, not values
- Makes action selection model-free too!



Reinforcement Learning -- Overview

- Passive Reinforcement Learning (= how to learn from experiences)
 - Model-based Passive RL
 - Learn the MDP model from experiences, then solve the MDP
 - Model-free Passive RL
 - Forego learning the MDP model, directly learn V or Q:
 - Value learning – learns value of a fixed policy; 2 approaches: Direct Evaluation & TD Learning
 - Q learning – learns Q values of the optimal policy (uses a Q version of TD Learning)
- Active Reinforcement Learning (= agent also needs to decide how to collect experiences)
 - Key challenges:
 - How to efficiently explore?
 - How to trade off exploration <> exploitation
 - Applies to both model-based and model-free.
we'll cover only in context of Q-learning

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:

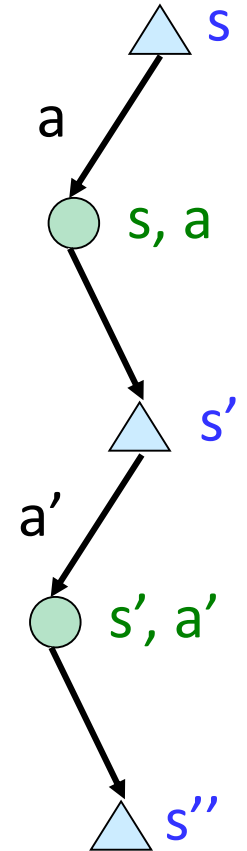
$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V_k(s') \right]$$

- 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:

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

Model-Free Learning

- Model-free (temporal difference) learning
 - Experience world through episodes
 $(s, a, r, s', a', r', s'', a'', r'', s'''' \dots)$
 - Update estimates each transition (s, a, r, s')
 - Over time, updates will mimic Bellman updates



Q-Learning

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

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

- Learn $Q(s,a)$ values as you go

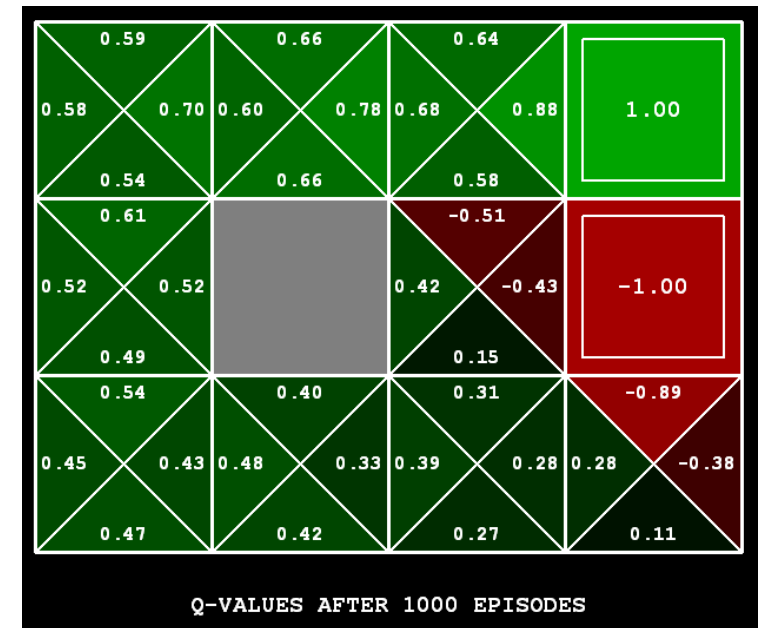
- Receive a sample (s,a,s',r)
- Consider your old estimate: $Q(s, a)$
- Consider your new sample estimate:

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

no longer policy evaluation!

- Incorporate the new estimate into a running average:

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

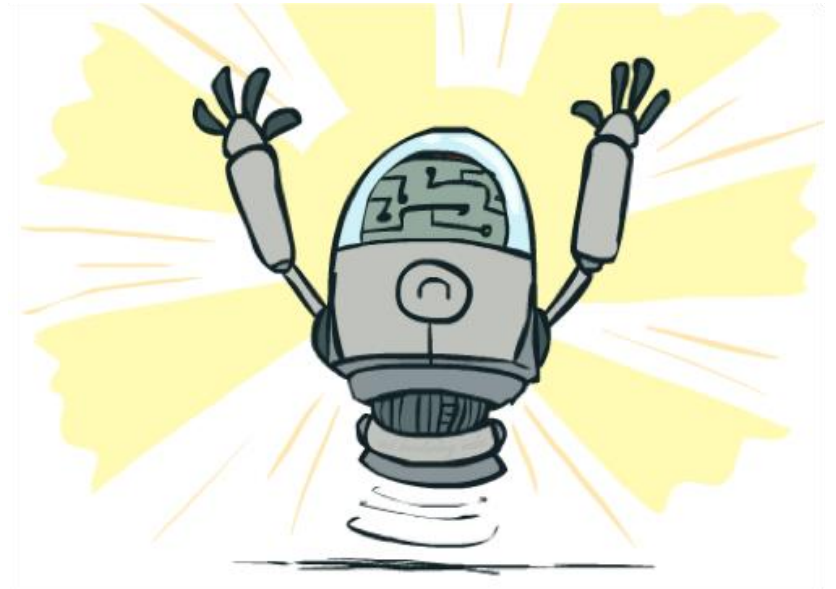


[Demo: Q-learning – gridworld (L10D2)]

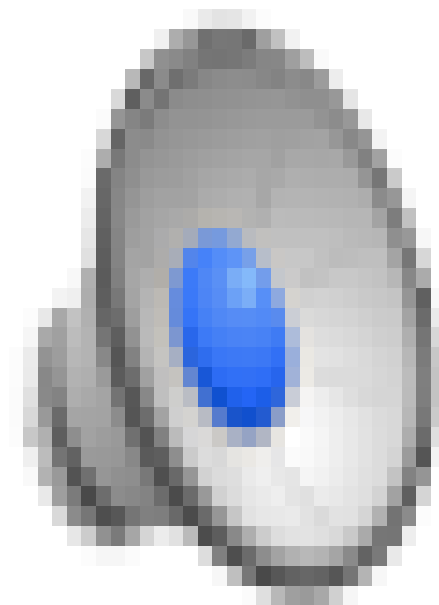
[Demo: Q-learning – crawler (L10D3)]

Q-Learning Properties

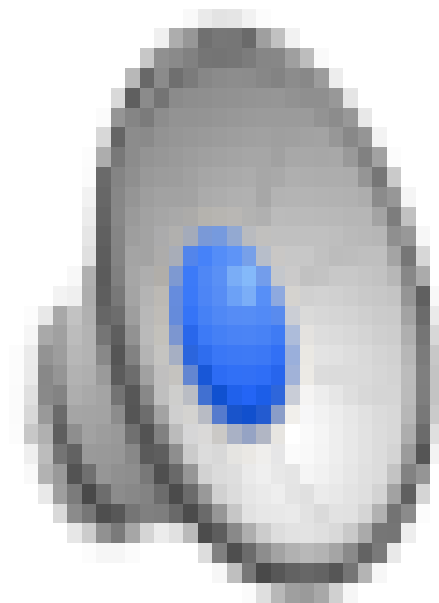
- Amazing result: Q-learning converges to optimal policy -- **even if you're acting suboptimally!**
- This is called **off-policy learning**
- Caveats:
 - You have to explore enough
 - You have to eventually make the learning rate small enough
 - ... but not decrease it too quickly
 - Basically, in the limit, it doesn't matter how you select actions (!)



Video of Demo Q-Learning -- Gridworld

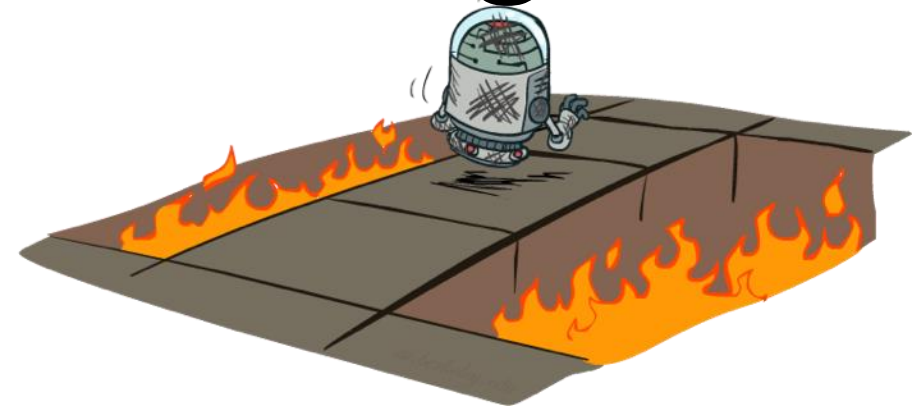


Video of Demo Q-Learning -- Crawler





Active Reinforcement Learning

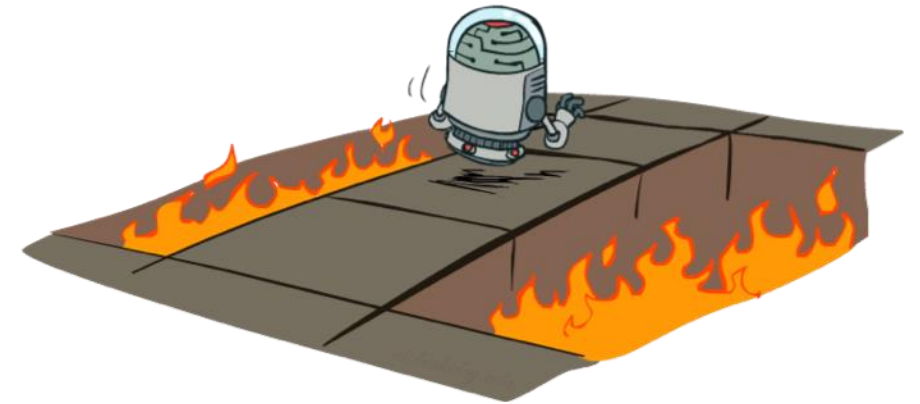


Reinforcement Learning -- Overview

- **Passive Reinforcement Learning (= how to learn from experiences)**
 - **Model-based Passive RL**
 - Learn the MDP model from experiences, then solve the MDP
 - **Model-free Passive RL**
 - Forego learning the MDP model, directly learn V or Q:
 - Value learning – learns value of a fixed policy; 2 approaches: Direct Evaluation & TD Learning
 - Q learning – learns Q values of the optimal policy (uses a Q version of TD Learning)
- **Active Reinforcement Learning (= agent also needs to decide how to collect experiences)**
 - **Key challenges:**
 - How to efficiently explore?
 - How to trade off exploration <> exploitation
 - Applies to both model-based and model-free.
we'll cover only in context of Q-learning

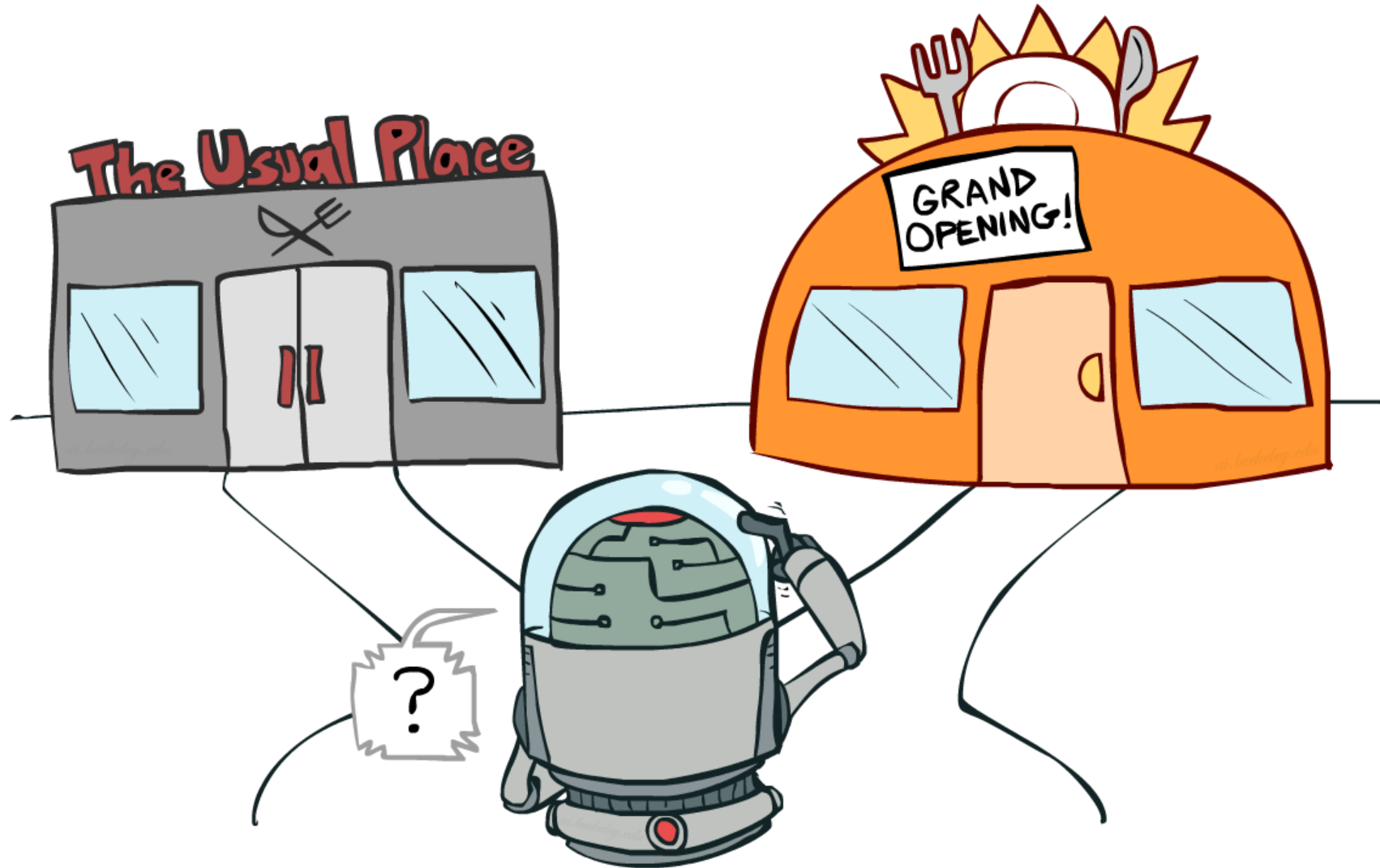
Active Reinforcement Learning

- 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

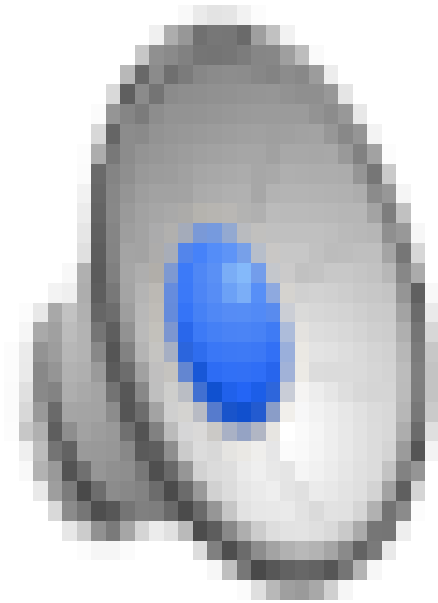


- In this case:
 - Learner makes choices!
 - Fundamental tradeoff: exploration vs. exploitation
 - This is NOT offline planning! You actually take actions in the world and find out what happens...

Exploration vs. Exploitation



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

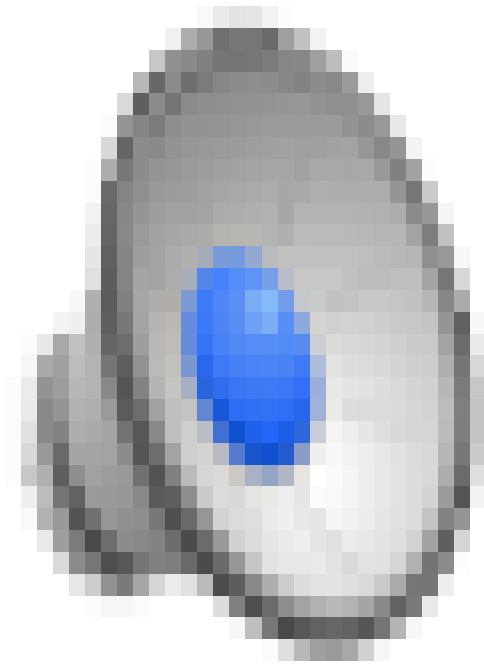


How to Explore?

- Several schemes for forcing exploration
 - Simplest: random actions (ϵ -greedy)
 - Every time step, flip a coin
 - With (small) probability ϵ , act randomly
 - With (large) probability $1-\epsilon$, 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



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



Exploration Functions

A commonly used ‘exploration function’ is $f(u, n) = u + c\sqrt{\log(1/\delta)/n}$, which is derived by Chernoff-Hoeffding inequality and δ is confidence level

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

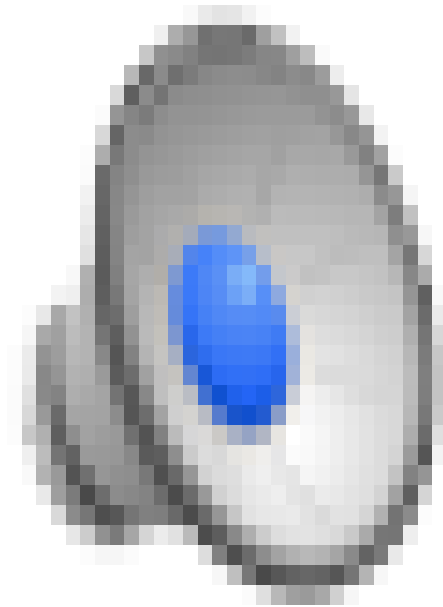
Regular Q-Update: $Q(s, a) \leftarrow_{\alpha} R(s, a, s') + \gamma \max_{a'} Q(s', a')$

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

- Action selection: Use $a \leftarrow \operatorname{argmax}_a Q(s, a)$

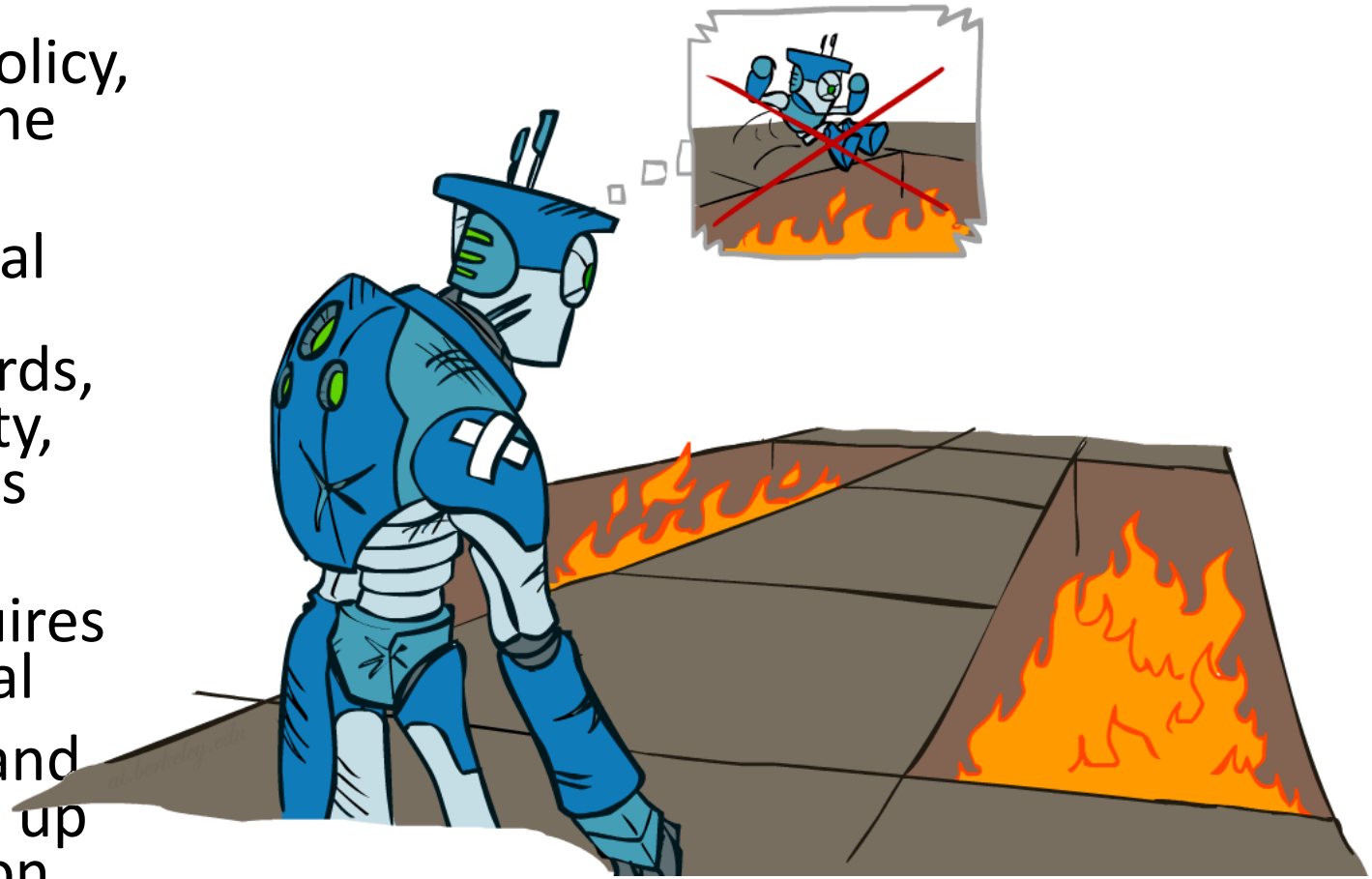
- Note: this propagates the “bonus” back to states that lead to unknown states as well!

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



Regret 2

- Cumulative regret, i.e., for episodic MDP with fixed horizon

$$R(T) = \sum_{t=1}^T (V^*(s_t) - V^{\pi_t}(s_t))$$

where s_t is the starting state of the t -th interaction game

- The algorithm is learning if the average regret converges, i.e.

$$\frac{R(T)}{T} \rightarrow 0, \text{ or equivalently } R(T) = o(T)$$

- Smaller order of $R(T)$ means faster learning speed
- Worst-case regret bound $R(T) = \Omega(\sqrt{T})$, which holds for a fixed game with arbitrary transitions and arbitrary (bounded) rewards

The Story So Far: MDPs and RL

Known MDP: Offline Solution

Goal

Compute V^*, Q^*, π^*

Evaluate a fixed policy π

Technique

Value / policy iteration

Policy evaluation

Unknown MDP: Model-Based

Goal

Compute V^*, Q^*, π^*

Evaluate a fixed policy π

Technique

VI/PI on approx. MDP

PE on approx. MDP

Unknown MDP: Model-Free

Goal

Compute V^*, Q^*, π^*

Evaluate a fixed policy π

Technique

Q-learning

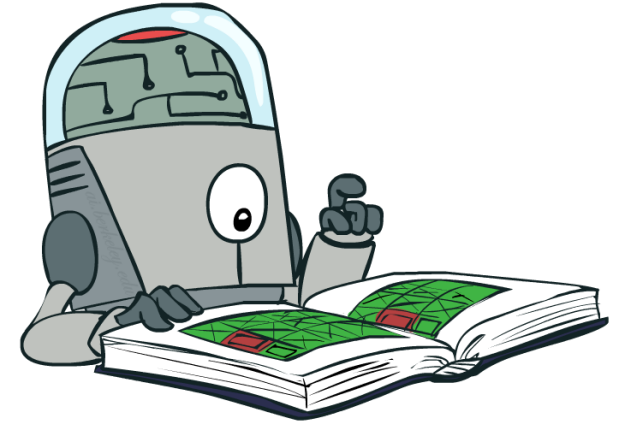
Value Learning

Reinforcement Learning -- Overview

- Passive Reinforcement Learning (= how to learn from experiences)
 - Model-based Passive RL
 - Learn the MDP model from experiences, then solve the MDP
 - Model-free Passive RL
 - Forego learning the MDP model, directly learn V or Q:
 - Value learning – learns value of a fixed policy; 2 approaches: Direct Evaluation & TD Learning
 - Q learning – learns Q values of the optimal policy (uses a Q version of TD Learning)
- Active Reinforcement Learning (= agent also needs to decide how to collect experiences)
 - Key challenges:
 - How to efficiently explore?
 - How to trade off exploration <> exploitation
 - Applies to both model-based and model-free.
we'll cover only in context of Q-learning
- Approximate Reinforcement Learning (= to handle large state spaces)
 - Approximate Q-Learning
 - Policy Search

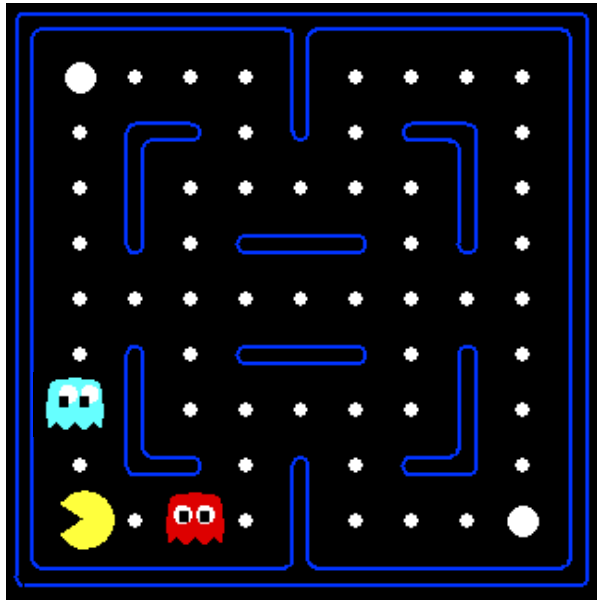
Generalizing Across States

- 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

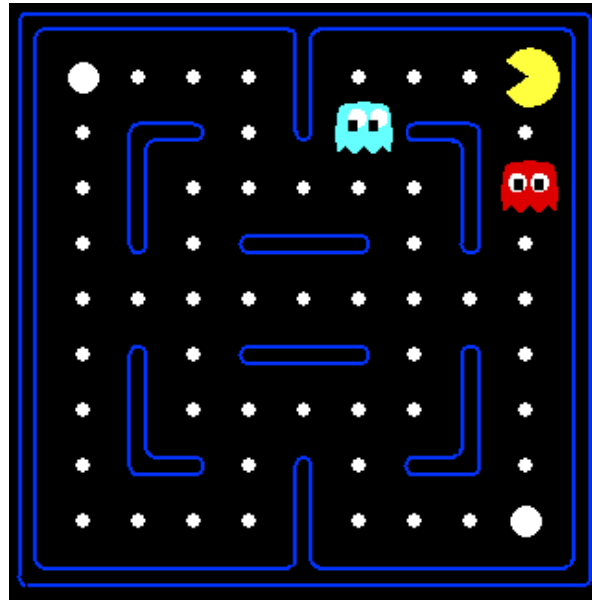


Example: Pacman

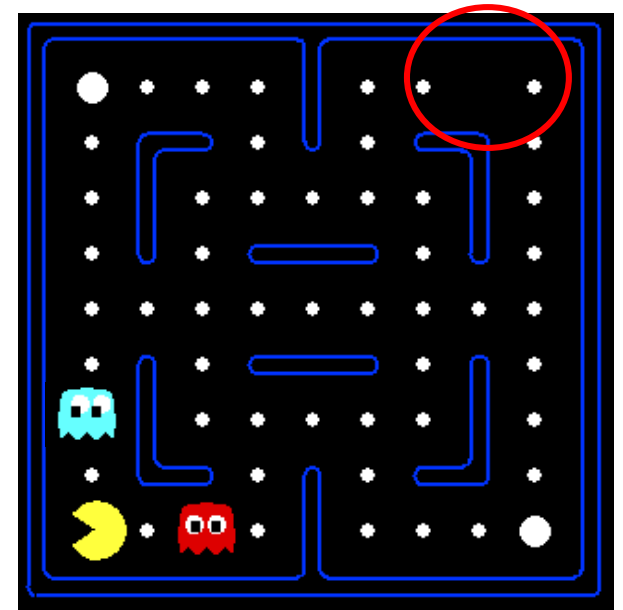
Let's say we discover through experience that this state is bad:



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

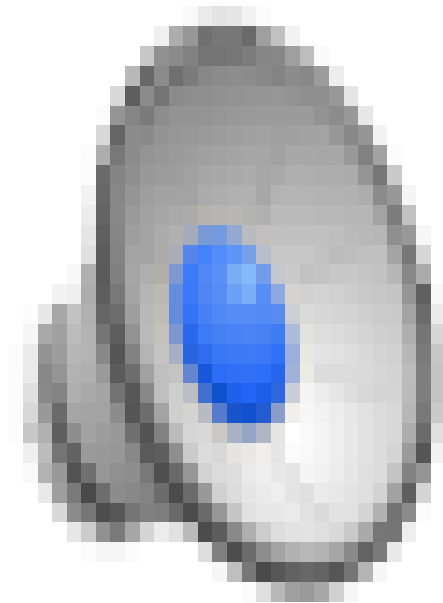


Or even this one!

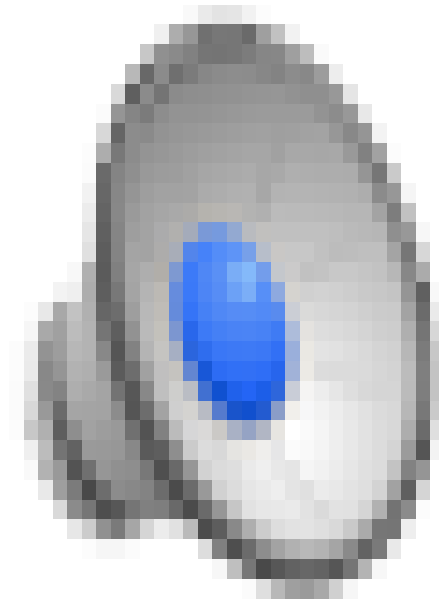


[Demo: Q-learning – pacman – tiny – watch all (L11D4)]
[Demo: Q-learning – pacman – tiny – silent train (L11D6)]
[Demo: Q-learning – pacman – tricky – watch all (L11D5)]

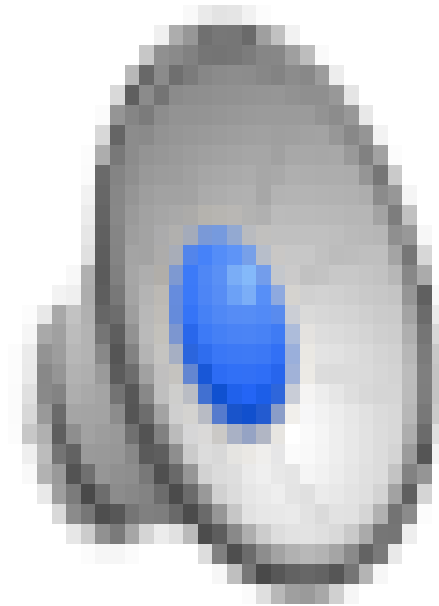
Video of Demo Q-Learning Pacman – Tiny – Watch All



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

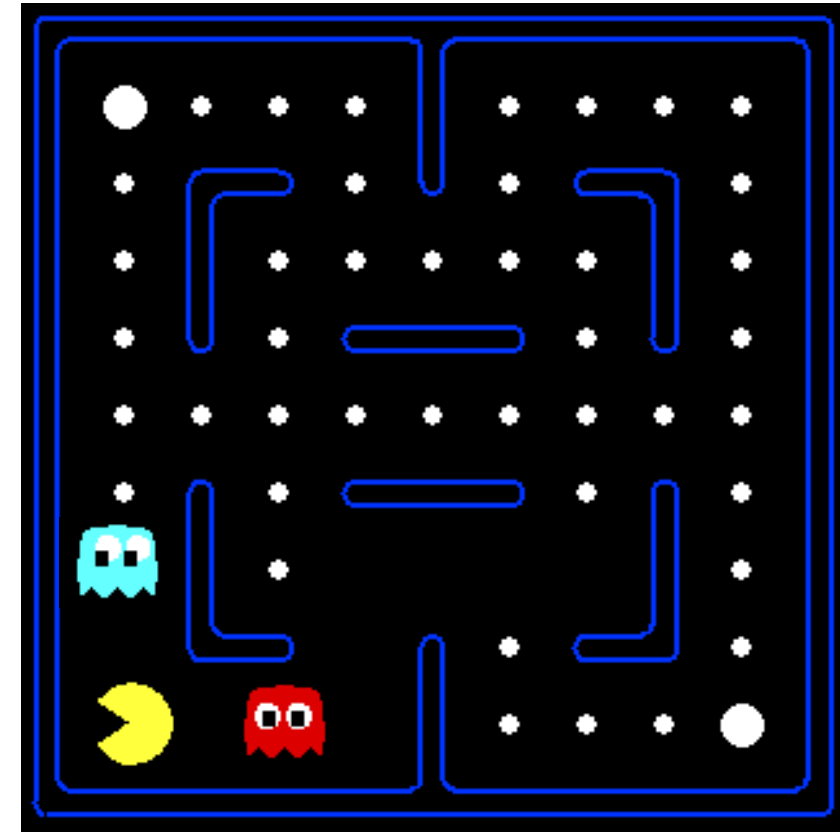


Video of Demo Q-Learning Pacman – Tricky – Watch All



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
 - $1 / (\text{dist to dot})^2$
 - 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:

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

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

- 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

$$Error(w) = \frac{1}{2} (\text{sample} - Q(s, a))^2$$
$$\frac{dError}{dw_i} = -(\text{sample} - Q(s, a)) f_i(s, a)$$

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

- Q-learning with linear Q-functions:

transition = (s, a, r, s')

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

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

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

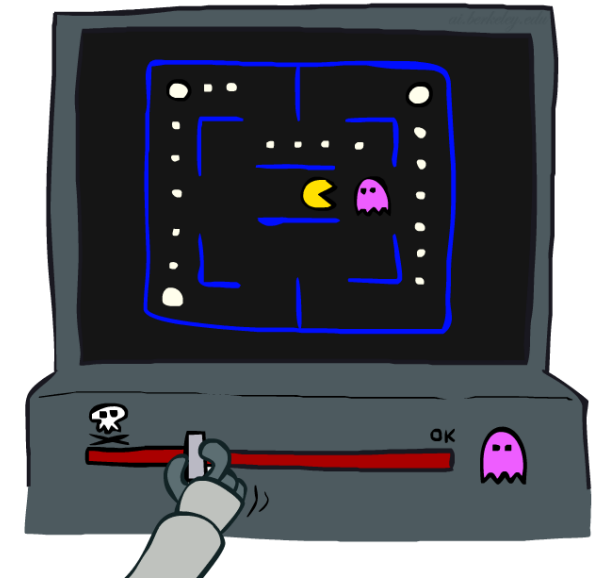
- Intuitive interpretation:

- 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

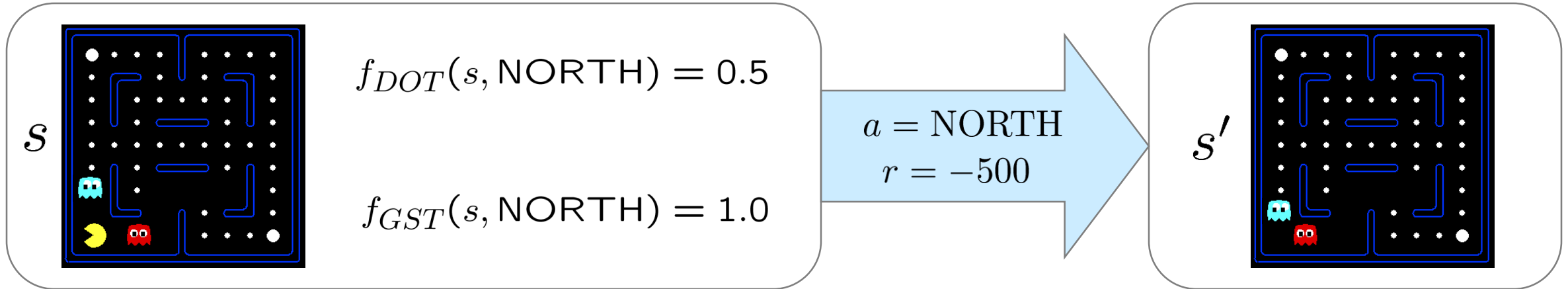
Exact Q's

Approximate Q's



Example: Q-Pacman

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



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

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

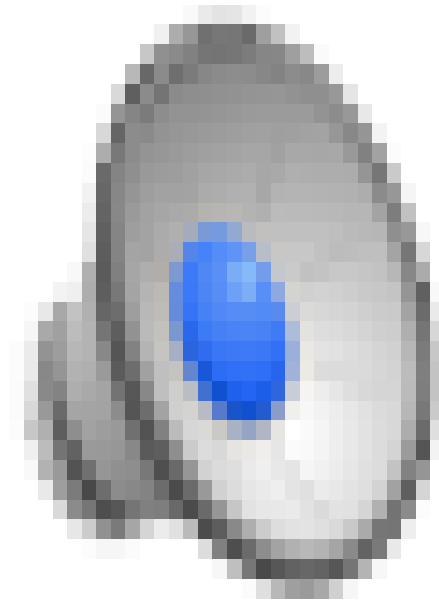
$$Q(s', \cdot) = 0$$

difference = -501 \longrightarrow

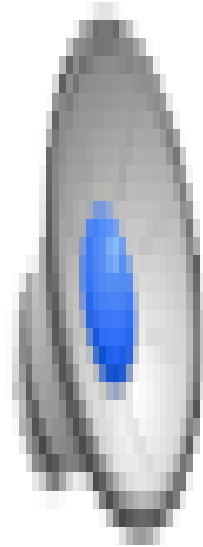
$$w_{DOT} \leftarrow 4.0 + \alpha [-501] 0.5$$
$$w_{GST} \leftarrow -1.0 + \alpha [-501] 1.0$$

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

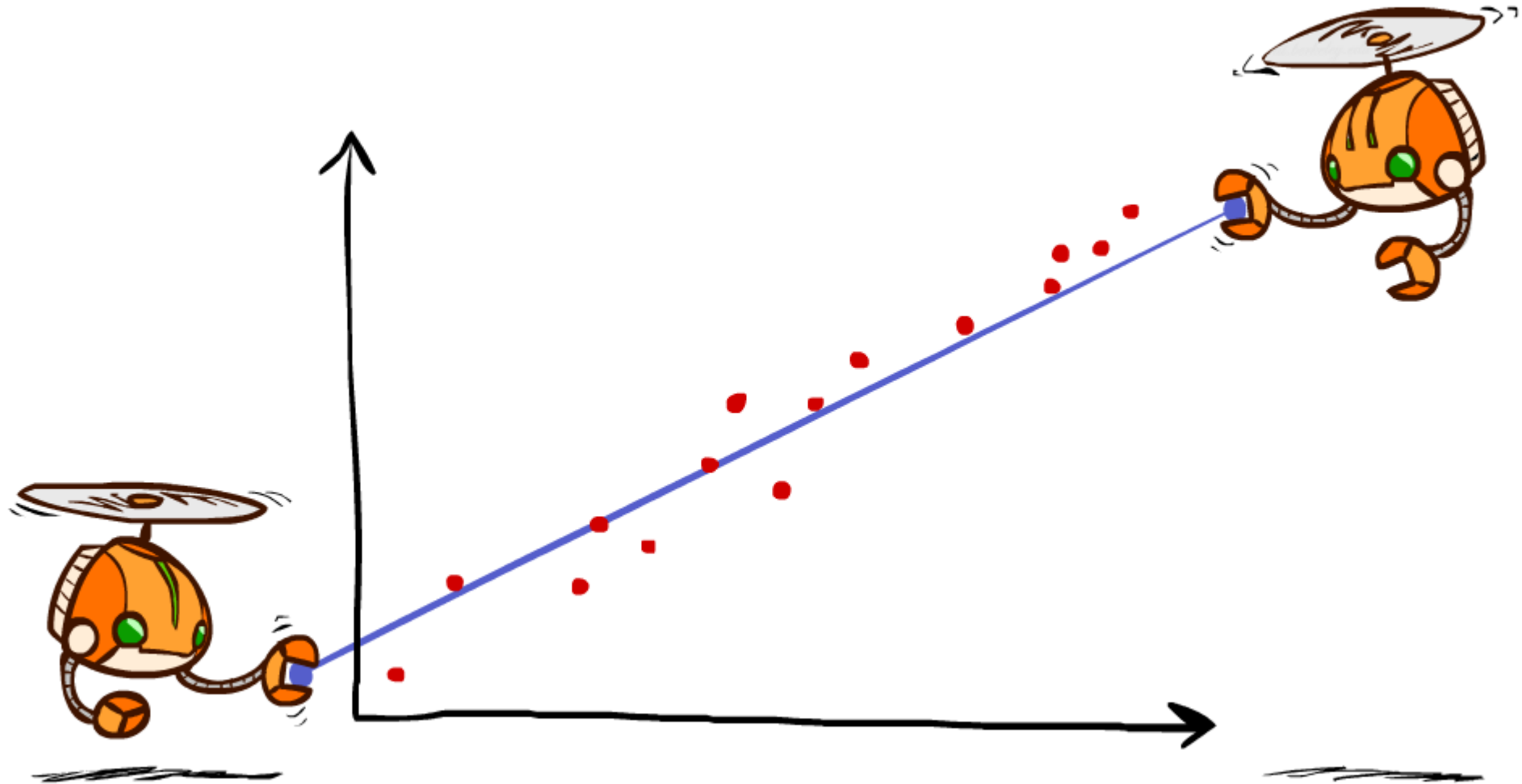
Video of Demo Approximate Q-Learning -- Pacman



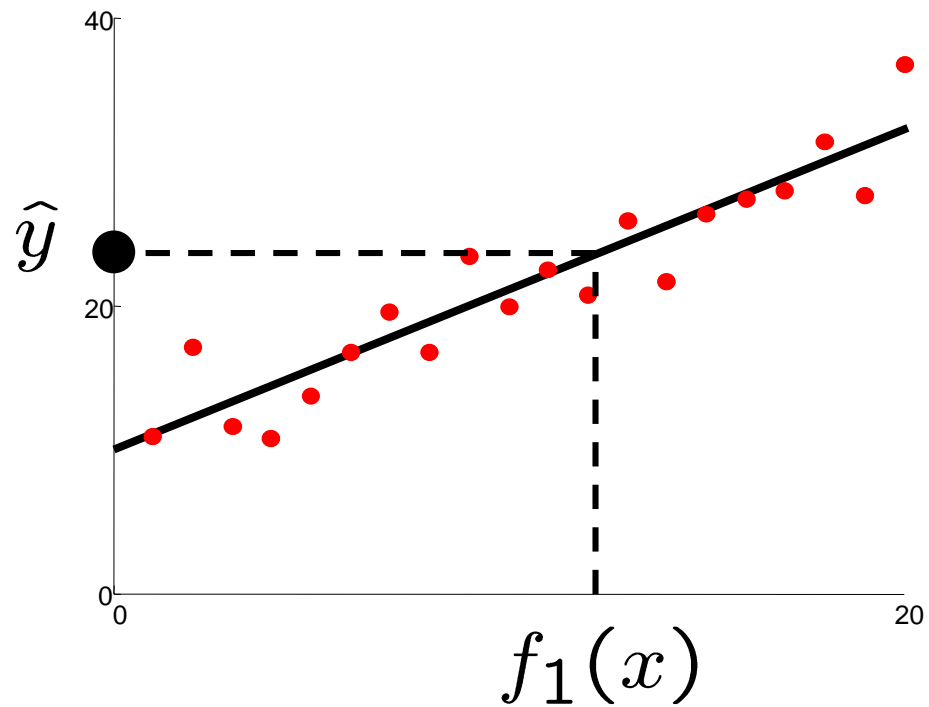
DeepMind Atari (©Two Minute Lectures) approximate Q-learning with neural nets



Q-Learning and Least Squares

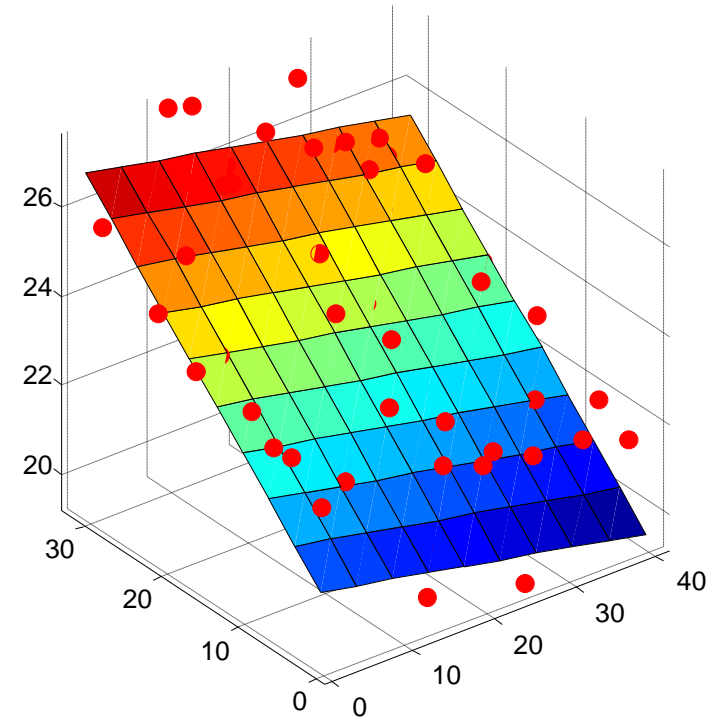


Linear Approximation: Regression



Prediction:

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

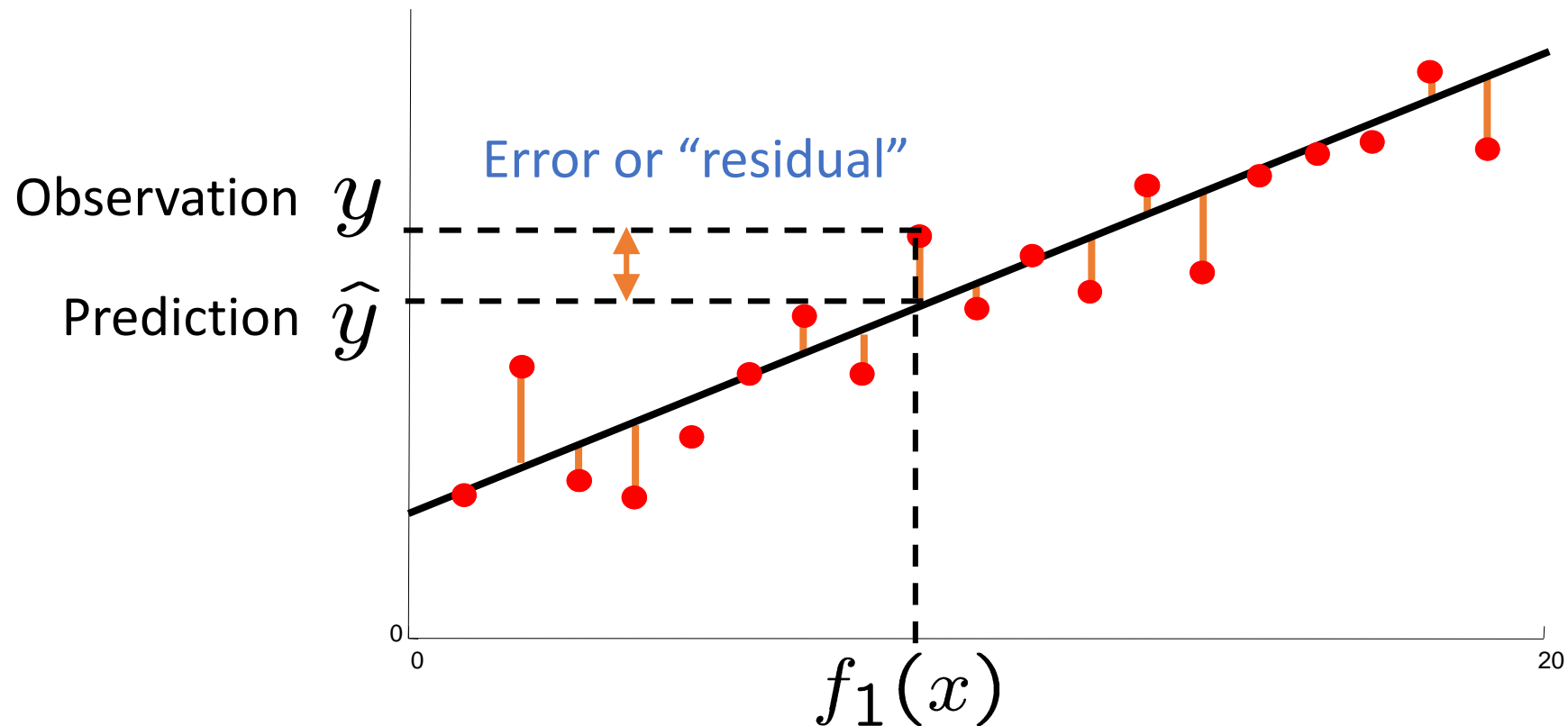


Prediction:

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

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 :

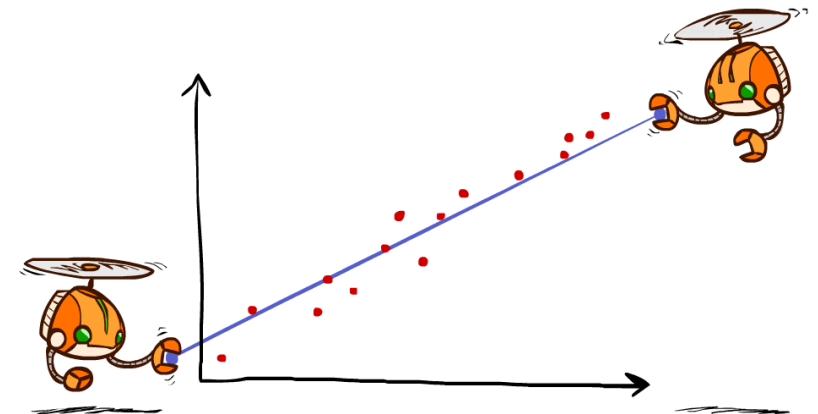
$$\text{error}(w) = \frac{1}{2} \left(y - \sum_k w_k f_k(x) \right)^2$$

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

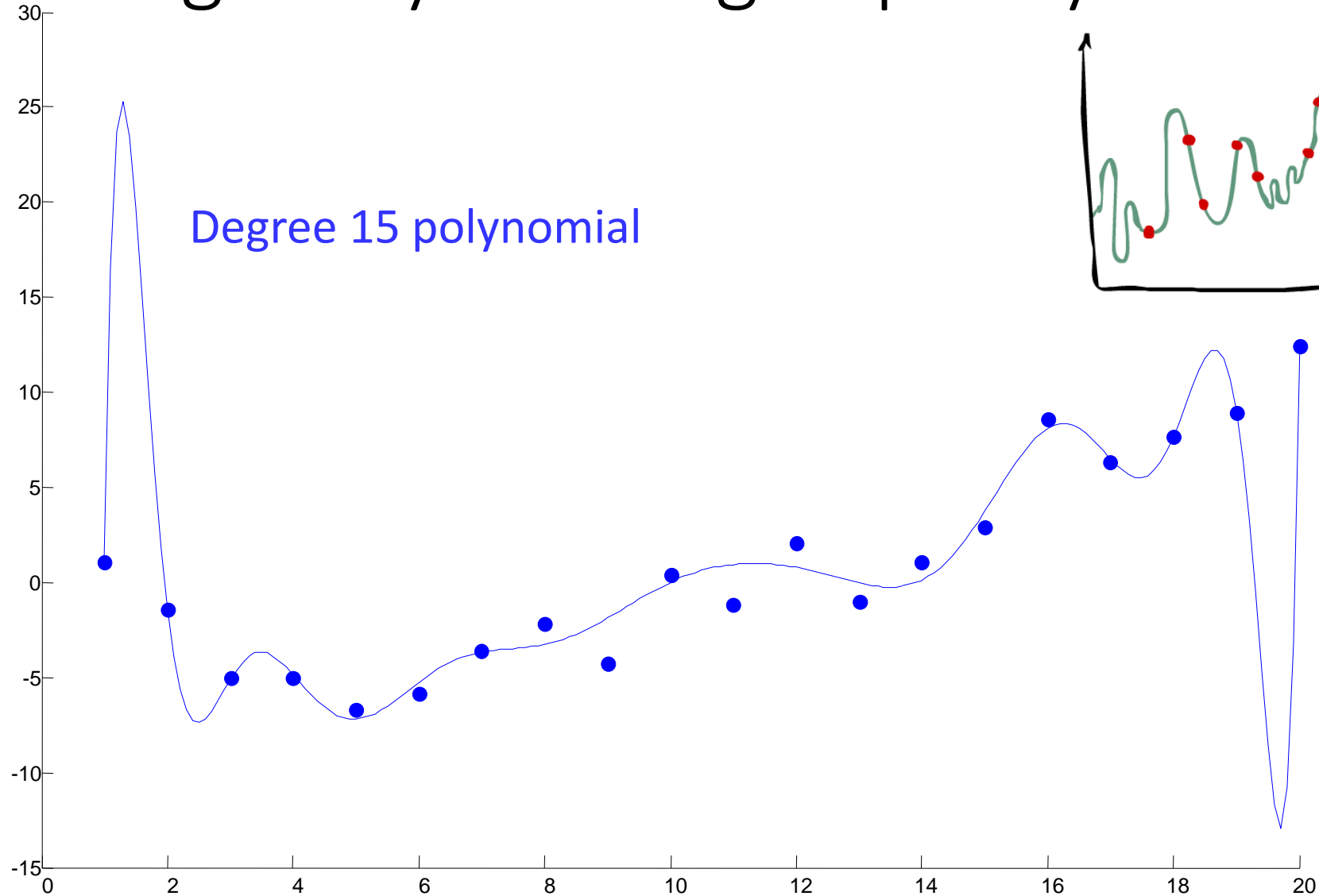
$$w_m \leftarrow w_m + \alpha \left(y - \sum_k w_k f_k(x) \right) f_m(x)$$

- Approximate q update explained:

$$w_m \leftarrow w_m + \alpha \left[\underbrace{r + \gamma \max_a Q(s', a')}_{\text{“target”}} - \underbrace{Q(s, a)}_{\text{“prediction”}} \right] f_m(s, a)$$

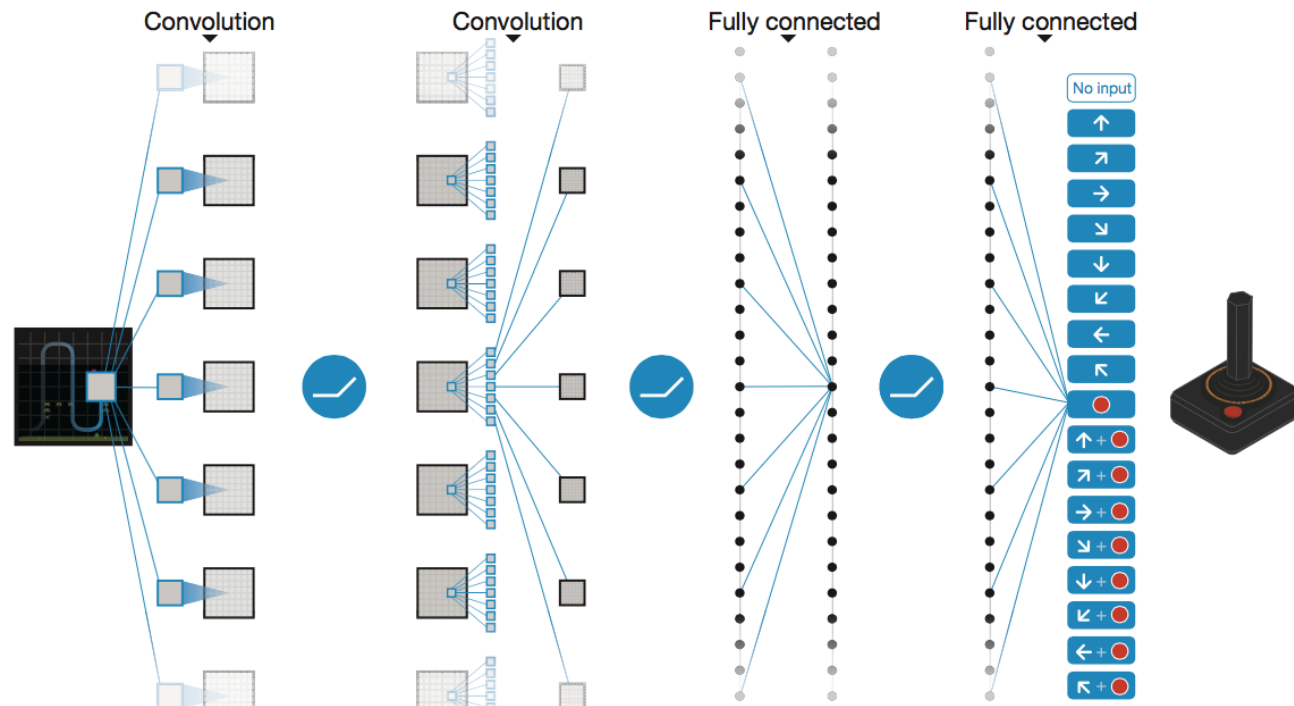


Overfitting: Why Limiting Capacity Can Help



Recent Advancements: Deep Q-Networks

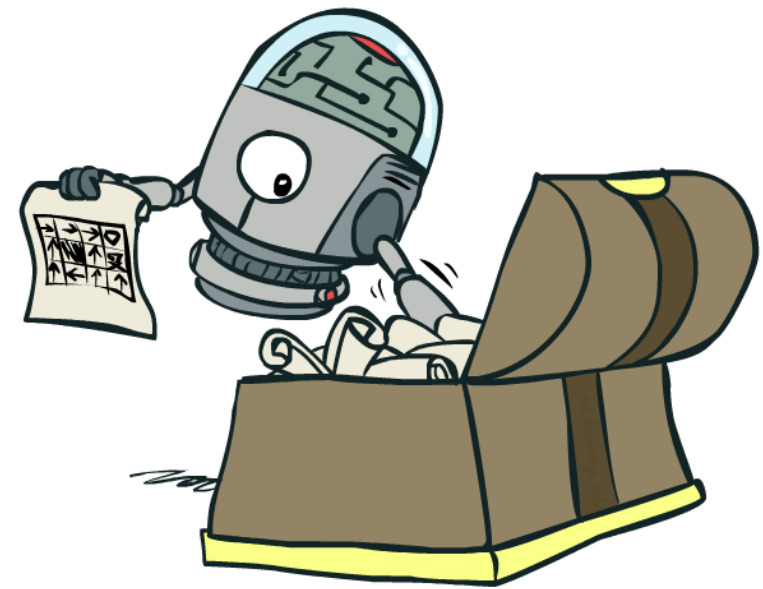
- Deep Mind, 2015
- Used a deep learning network to represent Q:
 - Input is last 4 images (84x84 pixel values) plus score
- 49 Atari games, incl. Breakout, Space Invaders, Seaquest, Enduro



Reinforcement Learning -- Overview

- Passive Reinforcement Learning (= how to learn from experiences)
 - Model-based Passive RL
 - Learn the MDP model from experiences, then solve the MDP
 - Model-free Passive RL
 - Forego learning the MDP model, directly learn V or Q:
 - Value learning – learns value of a fixed policy; 2 approaches: Direct Evaluation & TD Learning
 - Q learning – learns Q values of the optimal policy (uses a Q version of TD Learning)
- Active Reinforcement Learning (= agent also needs to decide how to collect experiences)
 - Key challenges:
 - How to efficiently explore?
 - How to trade off exploration <> exploitation
 - Applies to both model-based and model-free.
we'll cover only in context of Q-learning
- Approximate Reinforcement Learning (= to handle large state spaces)
 - Approximate Q-Learning
 - 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
 - E.g. some value functions have 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)
 - 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 2

- 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...

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

Unknown MDP: Model-Free

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

Summary

Shuai Li

<https://shuaili8.github.io>

- Passive Reinforcement Learning (= how to learn from experiences)
 - Model-based Passive RL
 - Model-free Passive RL
 - Direct Evaluation & TD Learning
 - Q learning
- Active Reinforcement Learning (= agent also needs to decide how to collect experiences)
 - Active Q-learning
 - Exploration vs Exploitation
- Approximate Reinforcement Learning (= to handle large state spaces)
 - Approximate Q-Learning
 - Policy Search

Questions?