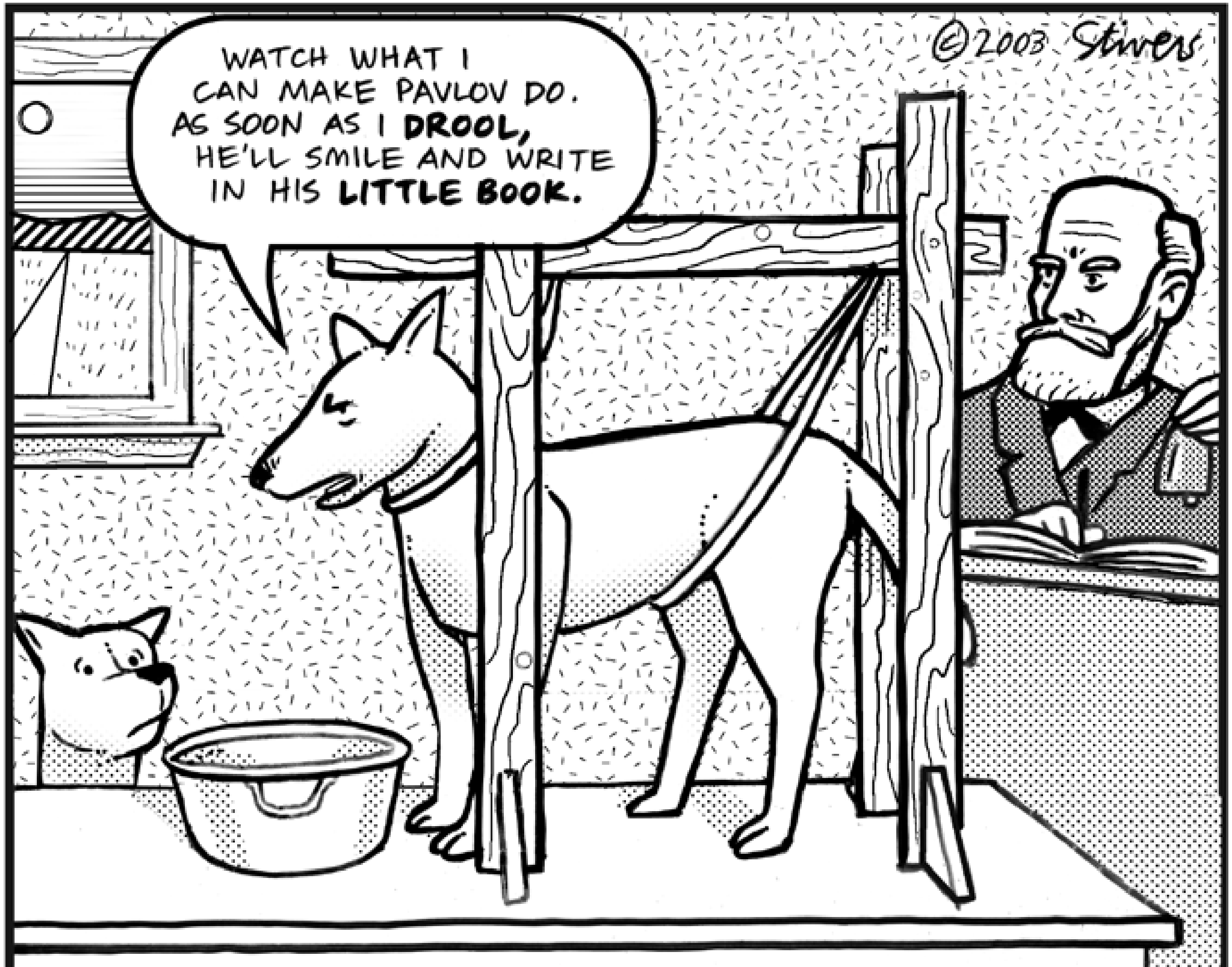
Reinforcement Learning

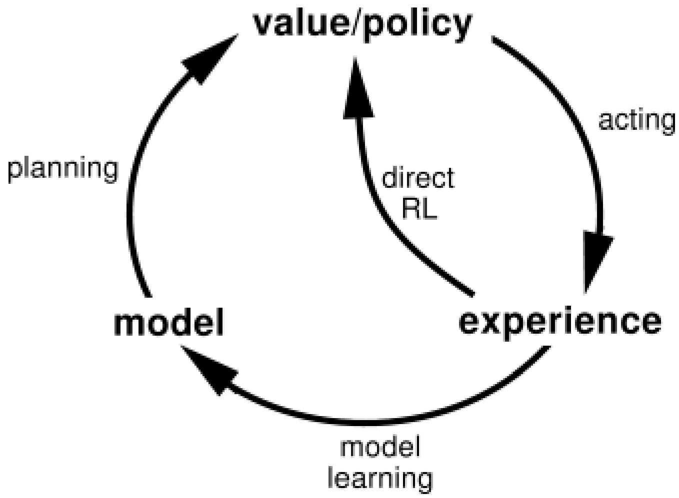
(Slides by Pieter Abbeel, Alan Fern,
Dan Klein, Subbarao Kambhampati,
Raj Rao, Lisa Torrey, Dan Weld)

[Many slides were taken from Dan Klein and Pieter Abbeel / CS188 Intro to AI at UC Berkeley. All CS188 materials are available at http://ai.berkeley.edu.]


WATCH WHAT I CAN MAKE PAVLOV DO. AS SOON AS I **DROOL,** HE'LL SMILE AND WRITE IN HIS **LITTLE BOOK.**
©2003 Stivers

- https://www.facebook.com/BiteesTreatsShow/videos/2073060332943406/

Learning/Planning/Acting



Reinforcement Learning

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

Main Dimensions

Model-based vs. Model-free

- Model-based vs. Model-free
  - Model-based → Have/learn action models (i.e. transition probabilities)
    - Eg. Approximate DP
  - Model-free → Skip them and directly learn what action to do when (without necessarily finding out the exact model of the action)
    - E.g. Q-learning

Passive vs. Active

- Passive vs. Active
  - Passive: Assume the agent is already following a policy (so there is no action choice to be made; you just need to learn the state values and may be action model)
  - Active: Need to learn both the optimal policy and the state values (and may be action model)

Main Dimensions (contd)

Extent of Backup

- Full DP
  - Adjust value based on values of *all* the neighbors (as predicted by the transition model)
  - Can only be done when transition model is present
- Temporal difference
  - Adjust value based only on the actual transitions observed

Strong or Weak Simulator

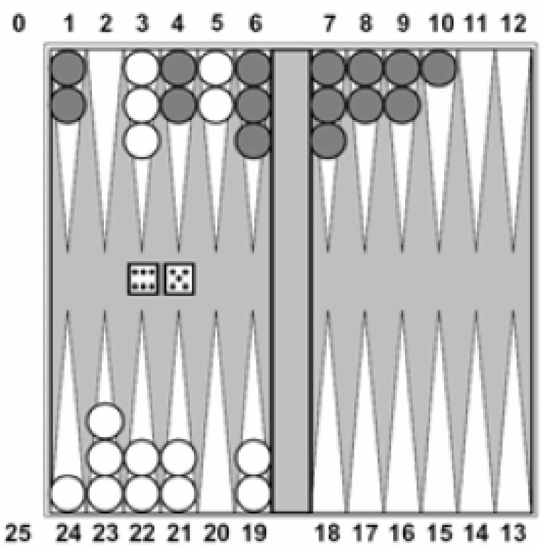
- Strong
  - I can jump to any part of the state space and start simulation there.
- Weak
  - Simulator is the real world and I can't teleport to a new state.

Example: Animal Learning

- RL studied experimentally for more than 60 years in psychology
  - Rewards: food, pain, hunger, drugs, etc.
  - Mechanisms and sophistication debated
- Example: foraging
  - Bees learn near-optimal foraging plan in field of artificial flowers with controlled nectar supplies
  - Bees have a direct neural connection from nectar intake measurement to motor planning area

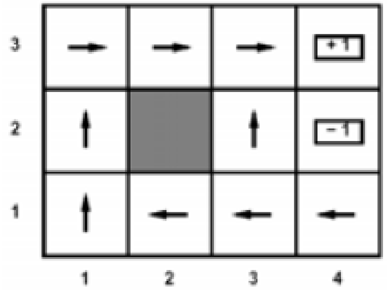
Example: Backgammon

- Reward only for win / loss in terminal states, zero otherwise
- TD-Gammon learns a function approximation to V(s) using a neural network
- Combined with depth 3 search, one of the top 3 players in the world



Does self learning through simulator.
[Infants don't get to "simulate" the world since they neither have T(.) nor R(.) of their world]

Passive Learning



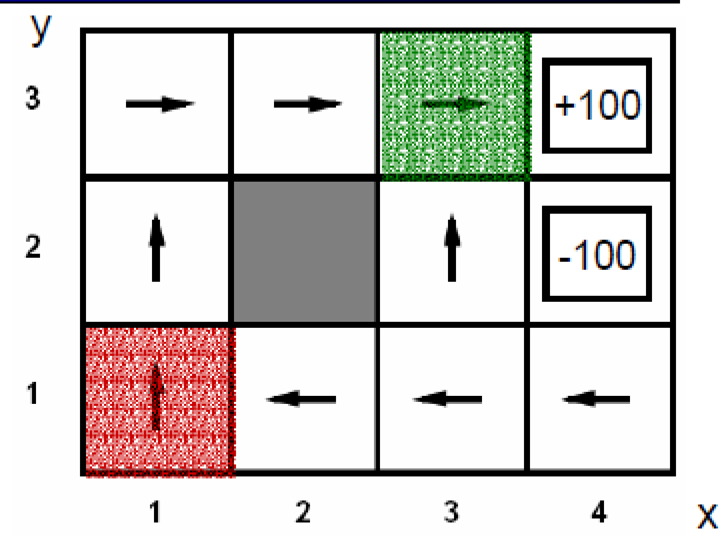
- Simplified task
  - You don't know the transitions T(s,a,s')
  - You don't know the rewards R(s,a,s')
  - You are given a policy $\pi(s)$
  - Goal: learn the state values (and maybe the model)
- In this case:
  - No choice about what actions to take
  - Just execute the policy and learn from experience
  - We'll get to the general case soon

We are basically doing EMPIRICAL Policy Evaluation!

Example: Direct Estimation

- Episodes:

(1,1) up -1	(1,1) up -1
(1,2) up -1	(1,2) up -1
(1,2) up -1	(1,3) right -1
(1,3) right -1	(2,3) right -1
(2,3) right -1	(3,3) right -1
(3,3) right -1	(3,2) up -1
(3,2) up -1	(4,2) exit -100
(3,3) right -1	(done)
(4,3) exit +100	
(done)	



$\gamma = 1$, $R = -1$

U(1,1) ~

U(3,3) ~

But we *know* this will be wasteful
(since it misses the correlation between values of neighboring states!)

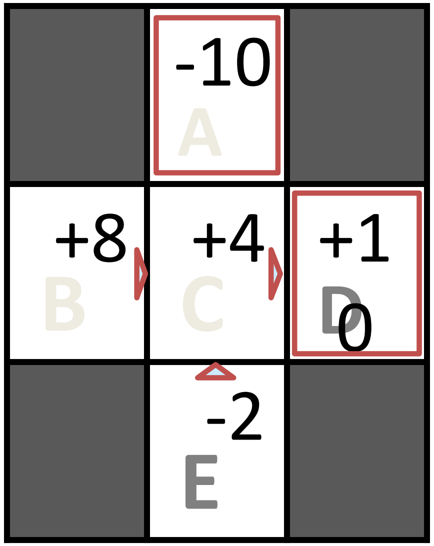
Do DP-based policy evaluation!

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
  - Ignores Bellman equations
  - 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?

Simple Example: Expected Age

Goal: Compute expected age of COL333 students

Known P(A)

[redacted] [redacted]

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

Unknown P(A): “Model Based”

[redacted]

[redacted]

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

Unknown P(A): “Model Free”

[redacted]

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

Model-Based Learning

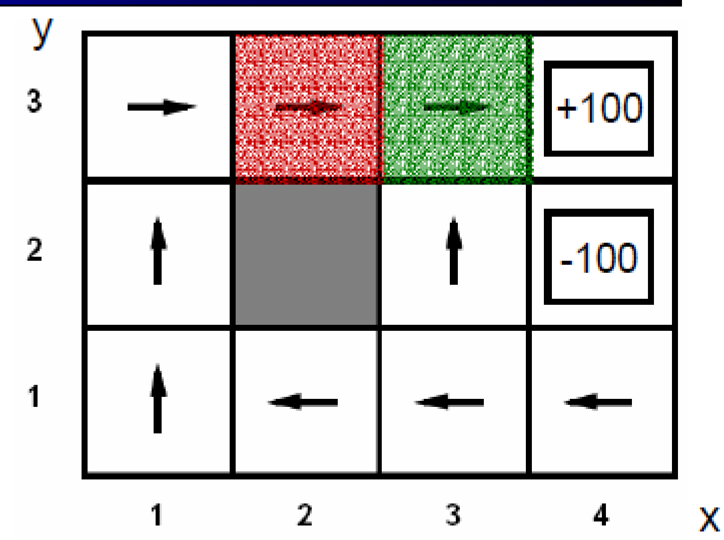
- Idea:
  - Learn the model empirically (rather than values)
  - Solve the MDP as if the learned model were correct
- Empirical model learning
  - Simplest case:
    - Count outcomes for each s,a
    - Normalize to give estimate of T(s,a,s')
    - Discover R(s,a,s') the first time we experience (s,a,s')
  - More complex learners are possible (e.g. if we know that all squares have related action outcomes, e.g. "stationary noise")

Example: Model-Based Learning

- Episodes:

(1,1) up -1
(1,2) up -1
(1,2) up -1
(1,3) right -1
(2,3) right -1
(3,3) right -1
(3,2) up -1
(3,3) right -1
(4,3) exit +100
(done)

(1,1) up -1
(1,2) up -1
(1,3) right -1
(2,3) right -1
(3,3) right -1
(3,2) up -1
(4,2) exit -100
(done)



$\gamma = 1$

T(<3,3>, right, <4,3>) = 1 / 3

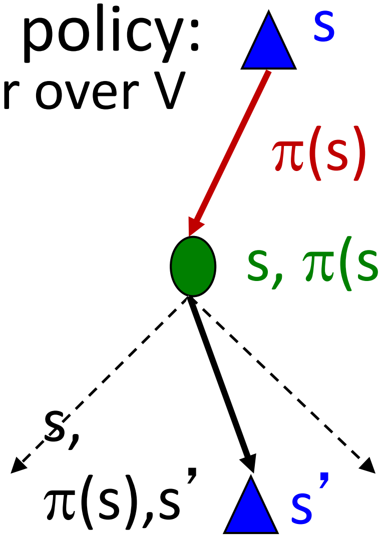
T(<2,3>, right, <3,3>) = 2 / 2

Model-based 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! (learn it -- model based)

- Key question: how can we do this update to V without knowing T and R? (model free)
  - 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:

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

- Idea: Take samples of outcomes s' action!) and average

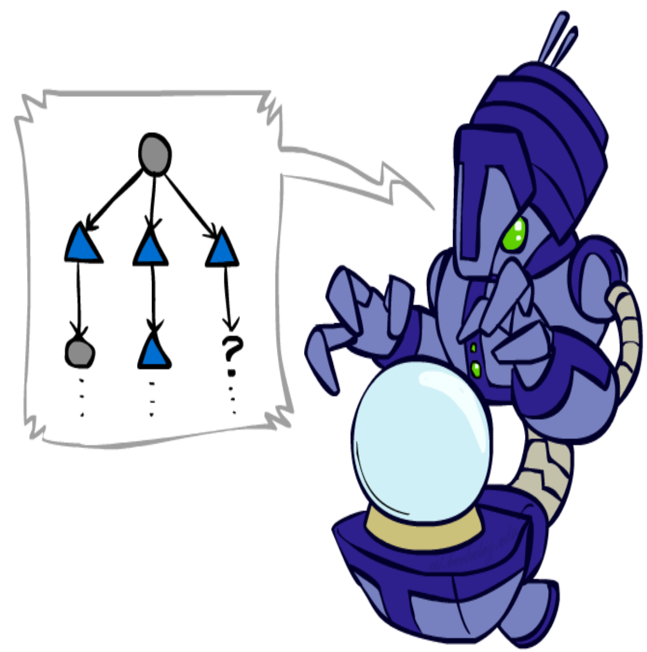
$$sample_1 = R(s, \pi(s), s_1') + \gamma V_k^{\pi}(s_1')$$

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

$$\dots$$

$$sample_n = R(s, \pi(s), s_n') + \gamma V_k^{\pi}(s_n')$$

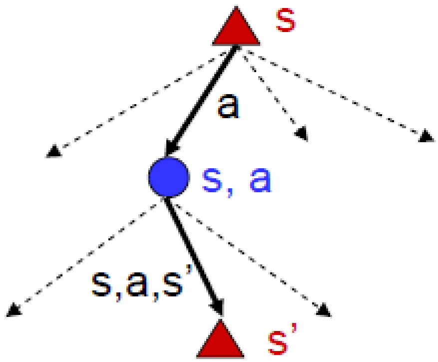
$$V_{k+1}^{\pi}(s) \leftarrow \frac{1}{n} \sum_i sample_i$$



sample from state

Model-Free Learning

- Big idea: why bother learning T?
  - Update each time we experience a transition
  - Frequent outcomes will contribute more updates (over time)
- Temporal difference learning (TD)
  - Policy still fixed!
  - Move values toward value of whatever successor occurs



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

$$sample = R(s, a, s') + \gamma V^\pi(s')$$

$$V^\pi(s) \leftarrow V^\pi(s) + \alpha(sample - V^\pi(s))$$

updated estimate learning rate

Aside: Online Mean Estimation

- Suppose that we want to incrementally compute the mean of a sequence of numbers ($x_{1,}$ $x_{2,}$ $x_{3,}$ ....)
  - E.g. to estimate the expected value of a random variable from a sequence of samples.

$$\hat{X}_{n+1} = \frac{1}{n+1} \sum_{i=1}^{n+1} x_i$$

average of n+1 samples

Aside: Online Mean Estimation

- Suppose that we want to incrementally compute the mean of a sequence of numbers (x_1, x_2, x_3, ….)
  - E.g. to estimate the expected value of a random variable from a sequence of samples.

$$\hat{X}_{n+1} = \frac{1}{n+1}\sum_{i=1}^{n+1} x_i = \frac{1}{n}\sum_{i=1}^{n} x_i + \frac{1}{n+1}\left(x_{n+1} - \frac{1}{n}\sum_{i=1}^{n} x_i \right)$$

average of n+1 samples

Aside: Online Mean Estimation

- Suppose that we want to incrementally compute the mean of a sequence of numbers ($x_1, x_2, x_3, \ldots$)
  - E.g. to estimate the expected value of a random variable from a sequence of samples.

$$\hat{X}_{n+1} = \frac{1}{n+1}\sum_{i=1}^{n+1} x_i = \frac{1}{n}\sum_{i=1}^{n} x_i + \frac{1}{n+1}\left(x_{n+1} - \frac{1}{n}\sum_{i=1}^{n} x_i\right)$$

$$= \hat{X}_n + \frac{1}{n+1}\left(x_{n+1} - \hat{X}_n\right)$$

average of n+1 samples

learning rate

sample n+1

- Given a new sample x_{n+1}, the new mean is the old estimate (for n samples) plus the weighted difference between the new sample and old estimate

Temporal Difference Learning

- TD update for transition from s to s':

$$V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha\big(R(s) + \gamma V^{\pi}(s') - V^{\pi}(s)\big)$$

updated estimate

learning rate

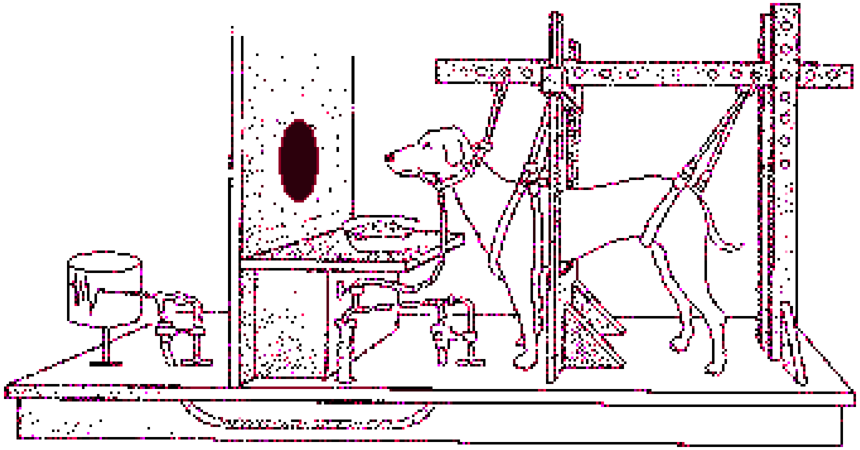
(noisy) sample of value at s based on next state s'

- So the update is maintaining a "mean" of the (noisy) value samples
- If the learning rate decreases appropriately with the number of samples (e.g. 1/n) then the value estimates will converge to true values! (non-trivial)

$$V^{\pi}(s) = R(s) + \gamma \sum_{s'} T(s,a,s') V^{\pi}(s')$$

Early Results: Pavlov and his Dog

- Classical (Pavlovian) conditioning experiments
- Training: Bell →Food
- After: Bell → Salivate
- Conditioned stimulus (bell) predicts future reward (food)



(http://employees.csbsju.edu/tcreed/pb/pdoganim.html)

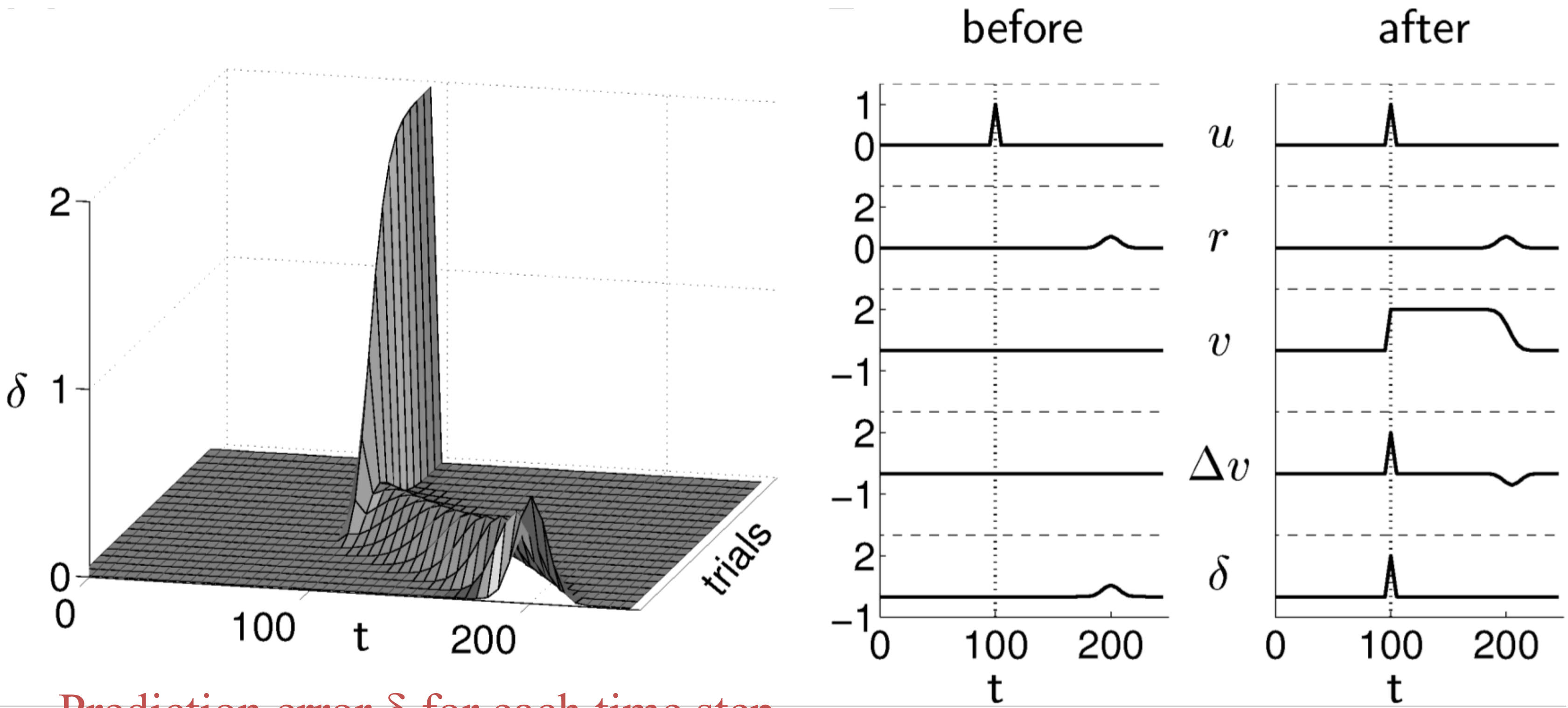
Predicting Delayed Rewards

- Reward is typically delivered at the end (when you know whether you succeeded or not)
- Time: $0 \le t \le T$ with stimulus $u(t)$ and reward $r(t)$ at each time step t (Note: $r(t)$ can be zero at some time points)
- Key Idea: Make the output $v(t)$ predict *total expected future reward* starting from time t

$$v(t) \approx \left\langle \sum_{\tau=0}^{T-t} r(t+\tau) \right\rangle$$

Predicting Delayed Reward: TD Learning

Stimulus at t = 100 and reward at t = 200

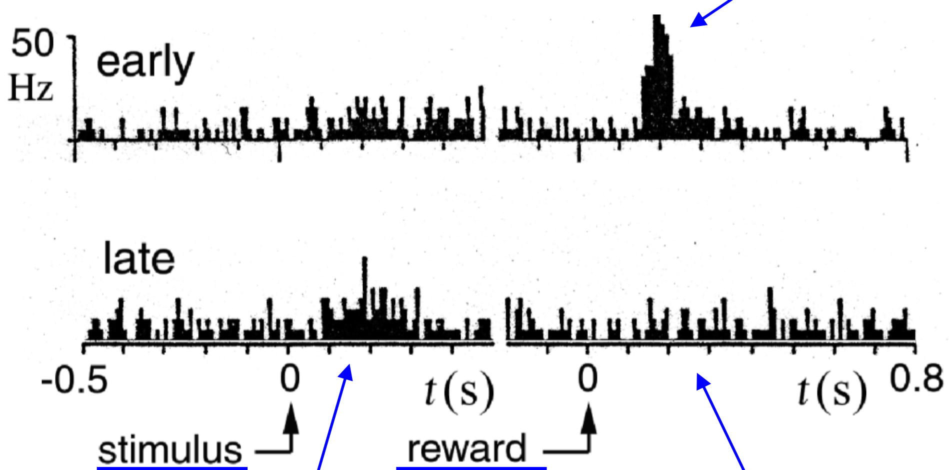


Prediction error δ for each time step
(over many trials)

Prediction Error in the Primate Brain?

Dopaminergic cells in Ventral Tegmental Area (VTA)

Reward Prediction error? $[r(t)+v(t+1)-v(t)]$



Before Training

After Training

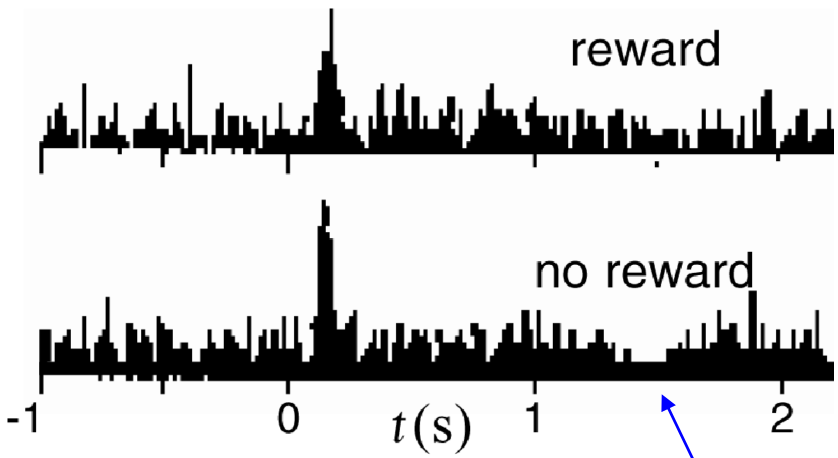
$[0+v(t+1)-v(t)]$

No error

$v(t)\approx r(t)+v(t+1)$

More Evidence for Prediction Error Signals

Dopaminergic cells in VTA



Negative error

$$r(t) = 0, v(t+1) = 0$$

$$[r(t) + v(t+1) - v(t)] = -v(t)$$

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

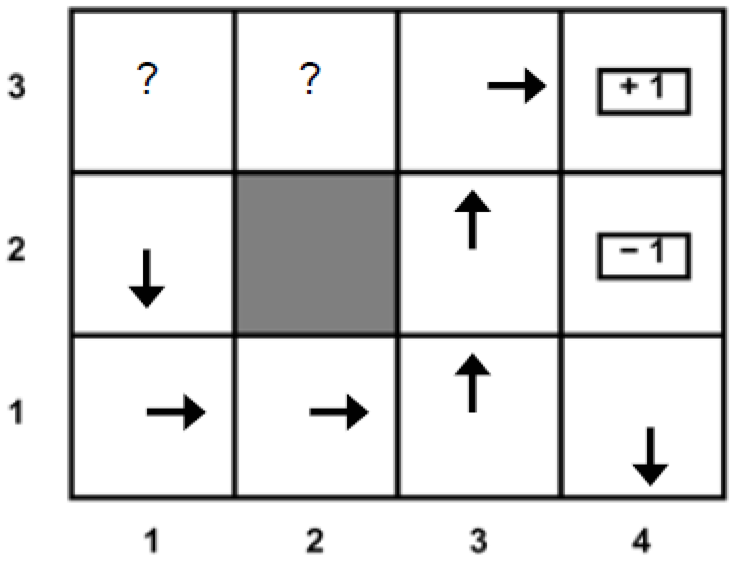
Goal	Technique
Compute V*, Q*, π*	Q-learning
Evaluate a fixed policy π	TD-Learning

Model-Based Learning

- In general, want to learn the optimal policy, not evaluate a fixed policy
- Idea: adaptive dynamic programming
  - Learn an initial model of the environment:
  - Solve for the optimal policy for this model (value or policy iteration)
  - Refine model through experience and repeat
  - Crucial: we have to make sure we actually learn about all of the model

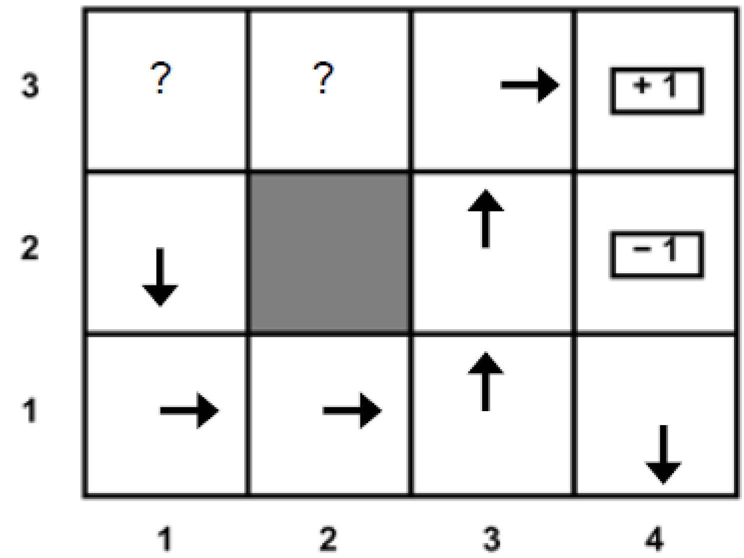
Example: Greedy ADP

- Imagine we find the lower path to the good exit first
- Some states will never be visited following this policy from (1,1)
- We'll keep re-using this policy because following it never collects the regions of the model we need to learn the optimal policy

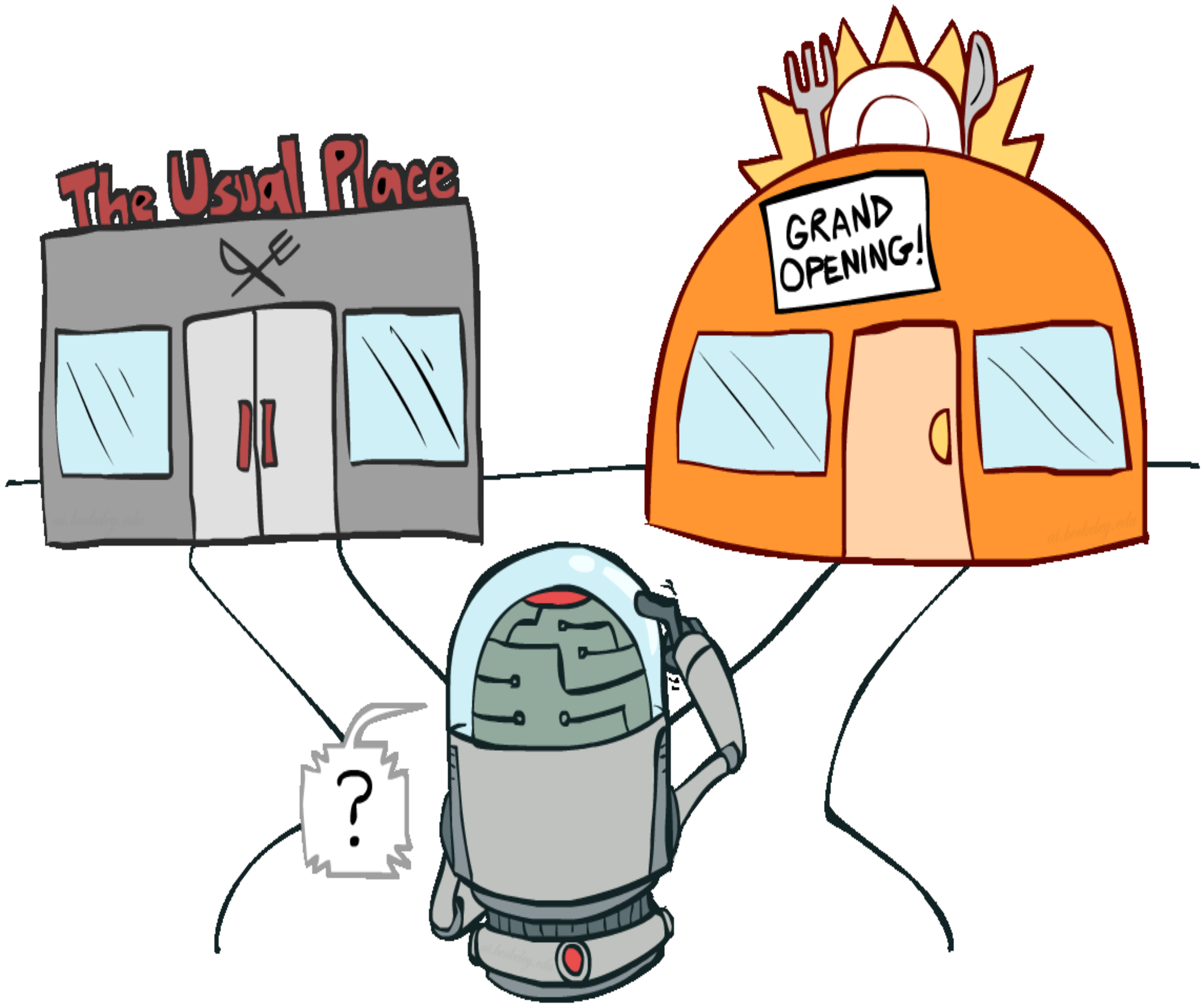


What Went Wrong?

- Problem with following optimal policy for current model:
  - Never learn about better regions of the space if current policy neglects them
- Fundamental tradeoff: exploration vs. exploitation
  - Exploration: must take actions with suboptimal estimates to discover new rewards and increase eventual utility
  - Exploitation: once the true optimal policy is learned, exploration reduces utility
  - Systems must explore in the beginning and exploit in the limit



Exploration vs. Exploitation



TD Learning → TD (V*) Learning

- Can we do TD-like updates on V*?
  - $V^*(s) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s')+\gamma V(s')]$
- Hmmm… what to do?

VI → Q-Value Iteration

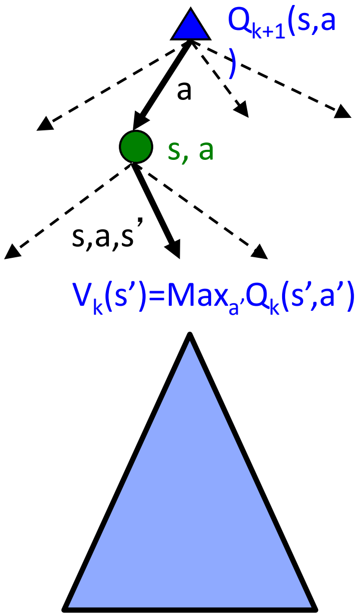
- **Forall s, a**
  - **Initialize $Q_0(s, a) = 0$** *no time steps left means an expected reward of zero*
- **K = 0**
- **Repeat** *do Bellman backups*

  For every (s,a) pair:

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

  K += 1

- **Until convergence** *I.e., Q values don't change much*



Q-Learning

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

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

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

  - Nudge the old estimate towards the new sample:

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

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

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

Q Learning

- **Forall s, a**
  - **Initialize Q(s, a) = 0**

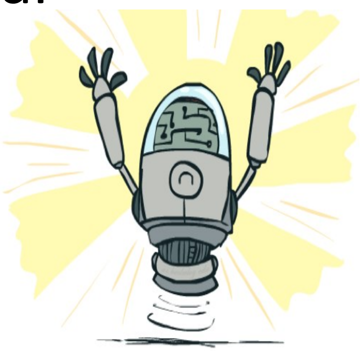
- **Repeat Forever**

  Where are you? s.
  Choose some action a
  Execute it in real world: *(s, a, r, s')*
  Do update:

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

Q-Learning Properties

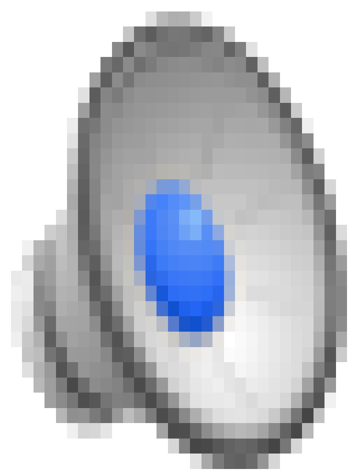
- Amazing result: Q-learning converges to optimal policy -- even if you're acting suboptimally!



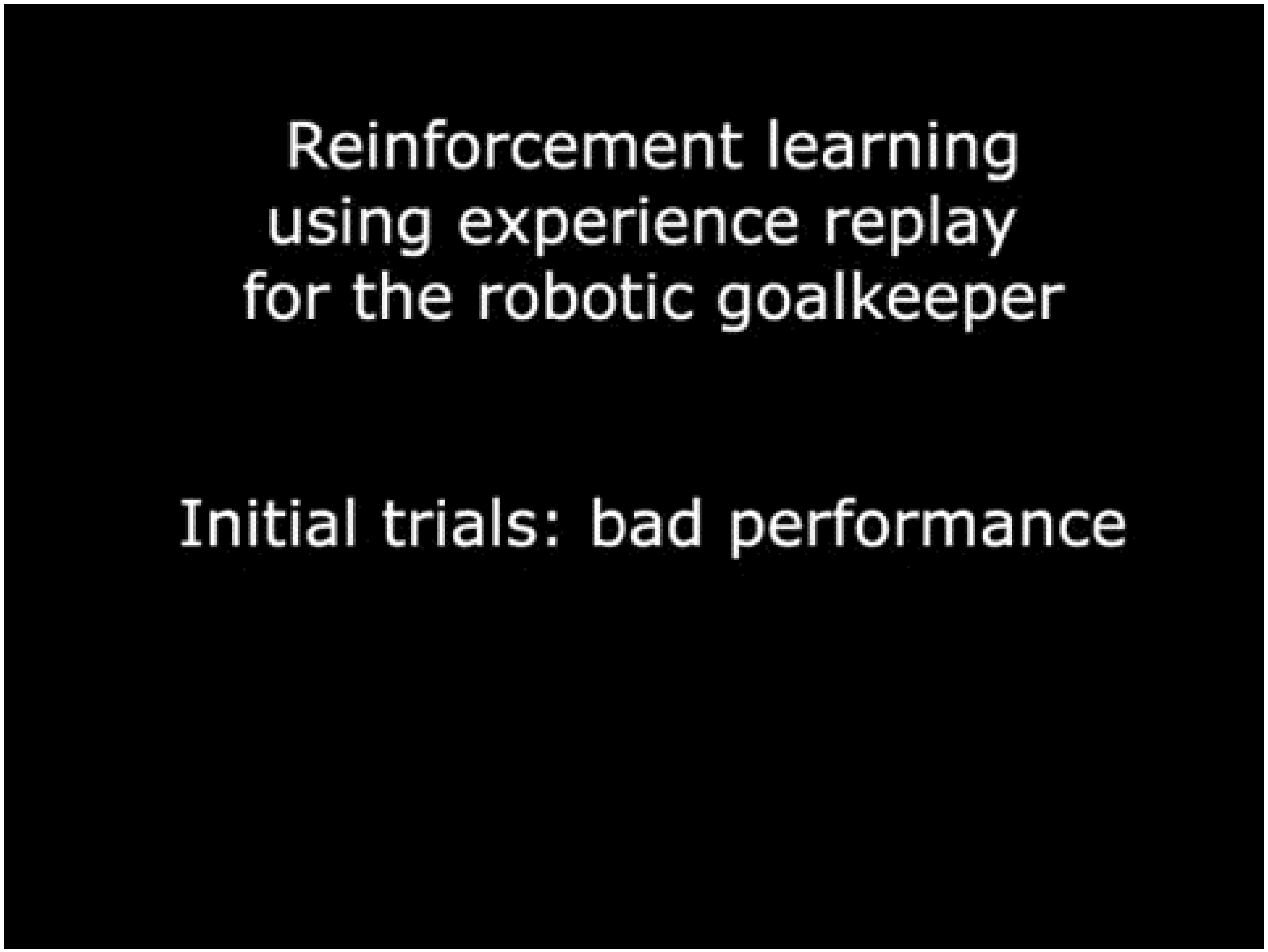
- This is called off-policy learning

- Caveats:
  - You have to explore enough
    - Exploration method would guarantee infinite visits to every state-action pair over an infinite training period
  - Learning rate decays with visits to state-action pairs
    - but not too fast decay. ($\sum_i \alpha(s,a,i) = \infty$, $\sum_i \alpha^2(s,a,i) < \infty$)
  - Basically, in the limit, it doesn't matter how you select actions (!)

Video of Demo Q-Learning Auto Cliff Grid

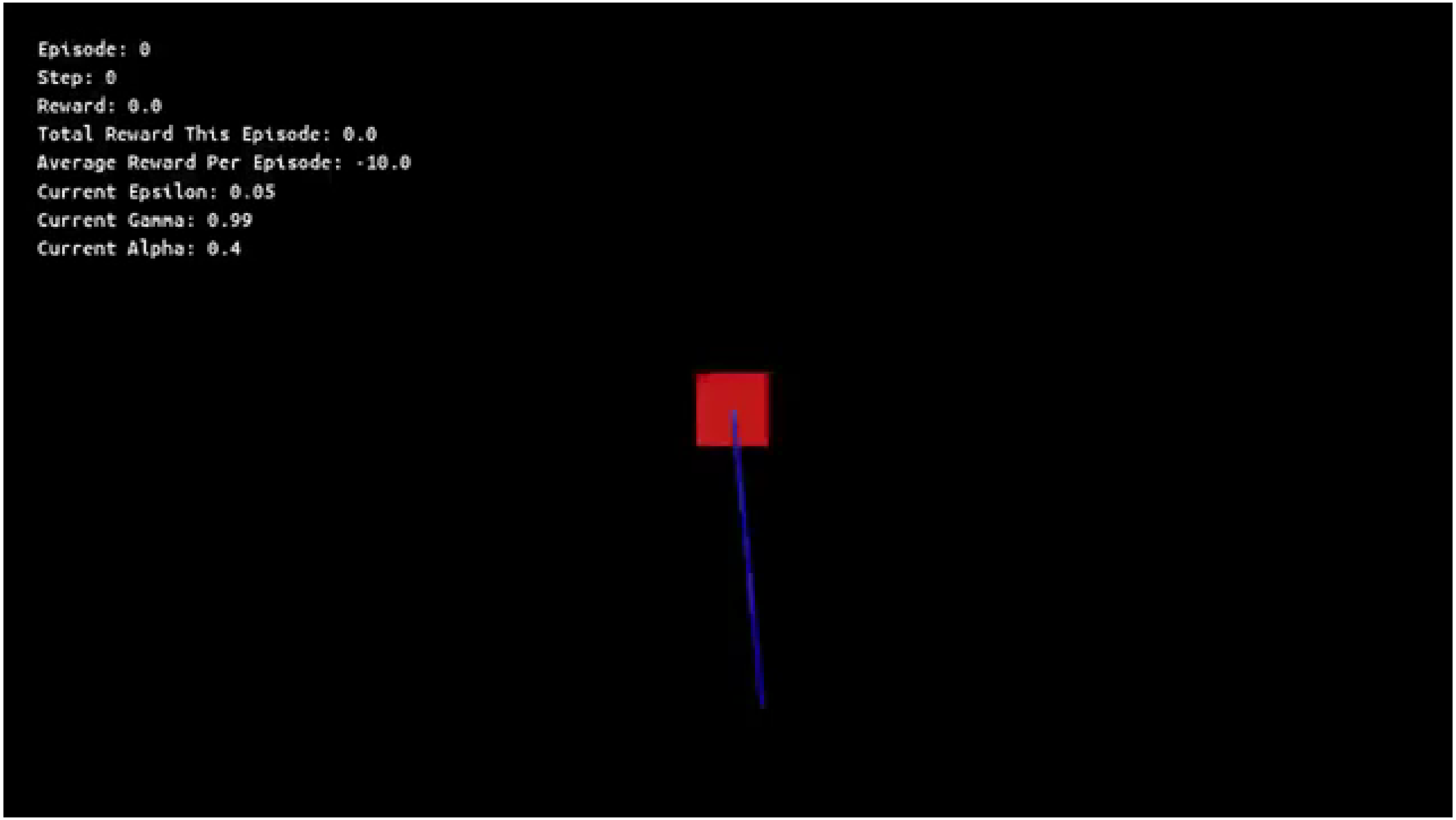


Example: Goalie



Video from [https://www.youtube.com/watch?v=CIF2SBVY-J0]

Example: Cart Balancing



[Video from https://www.youtube.com/watch?v=_Mmc3i7jZ2c]

Q-Learning Properties

- Will converge to optimal policy
  - If you explore enough
  - If you make the learning rate small enough
- Under certain conditions:
  - The environment model doesn't change
  - States and actions are finite
  - Rewards are bounded
  - Learning rate decays with visits to state-action pairs
    - but not too fast decay. ($\sum_i \alpha(s,a,i) = \infty$, $\sum_i \alpha^2(s,a,i) < \infty$)
  - Exploration method would guarantee infinite visits to every state-action pair over an infinite training period

Q Learning

- **Forall s, a**
  - **Initialize Q(s, a) = 0**

- **Repeat Forever**

  Where are you? s.

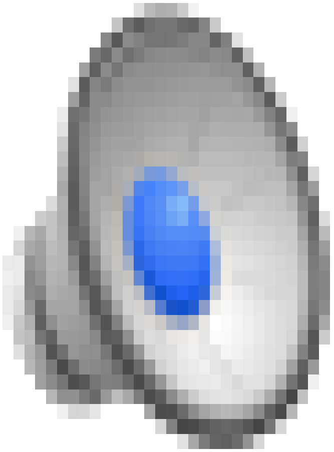
  Choose some action a

  Execute it in real world: *(s, a, r, s')*

  Do update:

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

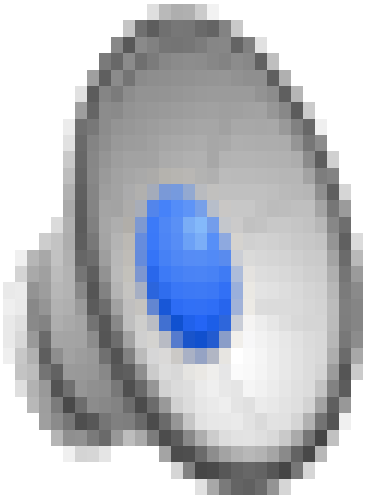
Video of Demo Q-learning – Manual Exploration – Bridge Grid



Exploration / Exploitation

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

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



Explore/Exploit Policies

- GLIE Policy 2: Boltzmann Exploration
  - Select action a with probability,

$$\Pr(a \mid s) = \frac{\exp\left(Q(s,a)/T\right)}{\sum_{a' \in A} \exp\left(Q(s,a')/T\right)}$$

  - T is the temperature. Large T means that each action has about the same probability. Small T leads to more greedy behavior.
  - Typically start with large T and decrease with time

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

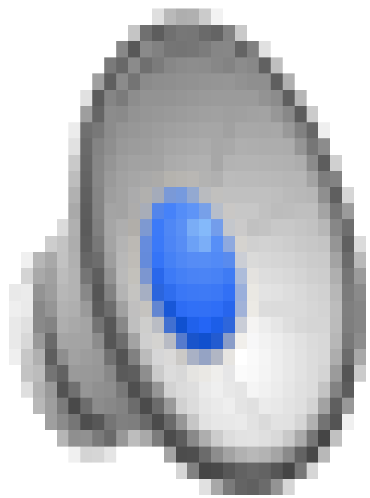
Regular Q-Update: $Q(s,a) \leftarrow_\alpha R(s,a,s') + \gamma \max_{a'} Q(s',a')$

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

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

[Demo: exploration – Q-learning – crawler – exploration function (L11D4)]

Video of Demo Q-learning – Exploration Function – Crawler

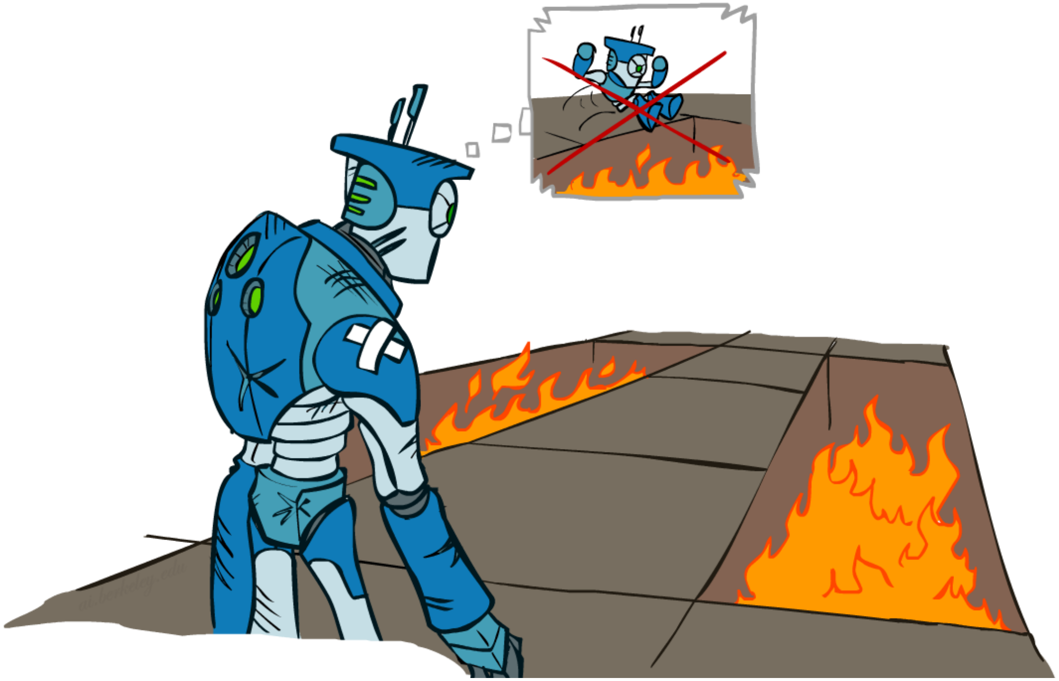


Model based vs. Model Free RL

- Model based
  - estimate O($|\mathcal{S}|^2|\mathcal{A}|$) parameters
  - requires relatively larger data for learning
  - can make use of background knowledge easily
- Model free
  - estimate O($|\mathcal{S}||\mathcal{A}|$) parameters
  - requires relatively less data for learning

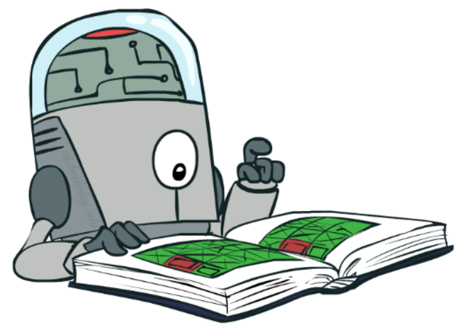
Regret

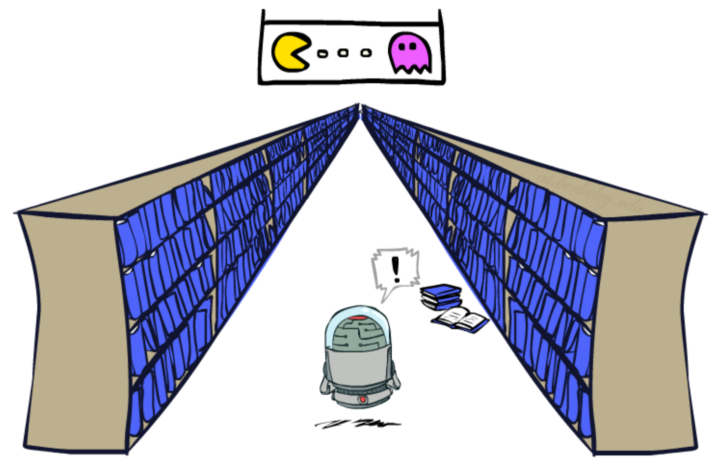
- Even if you learn the optimal policy, y
still make mistakes along the way!
- Regret is a measure of your total mis
cost: the difference between your
(expected) rewards, including youthf
suboptimality, and optimal (expectec
rewards
- Minimizing regret goes beyond learn
to be optimal – it requires optimally
learning to be optimal
- Example: random exploration and
exploration functions both end up
optimal, but random exploration has
higher regret



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

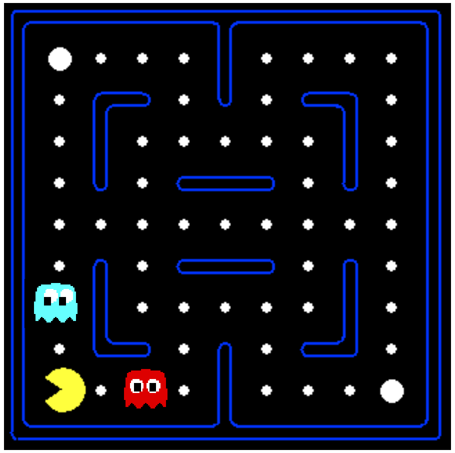




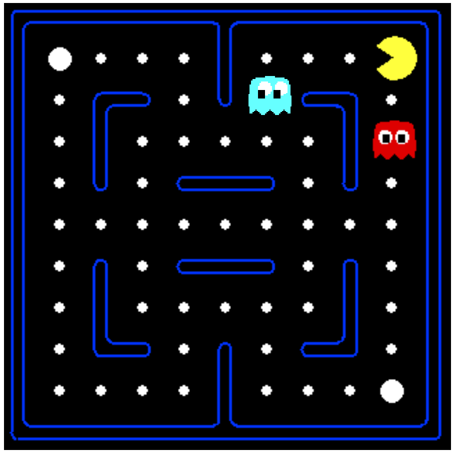
[demo – RL pacman]

Example: Pacman

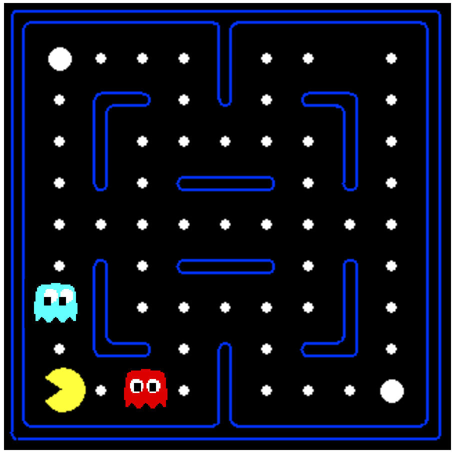
Let's say we discover through experience that this state is bad:



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

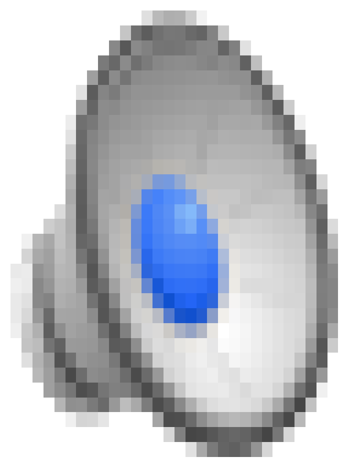


Or even this one!

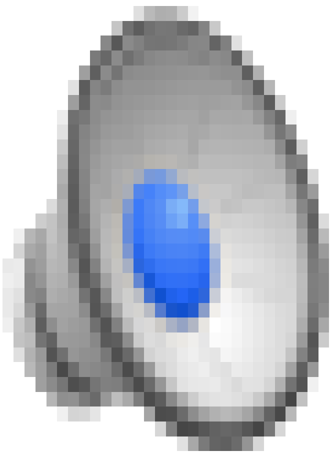


[Demo: Q-learning – pacman – tiny – watch all (L11D5)]
[Demo: Q-learning – pacman – tiny – silent train (L11D6)]
[Demo: Q-learning – pacman – tricky – watch all (L11D7)]

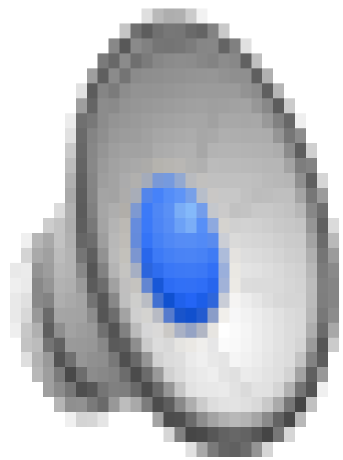
Video of Demo Q-Learning Pacman – Tiny – Watch All



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

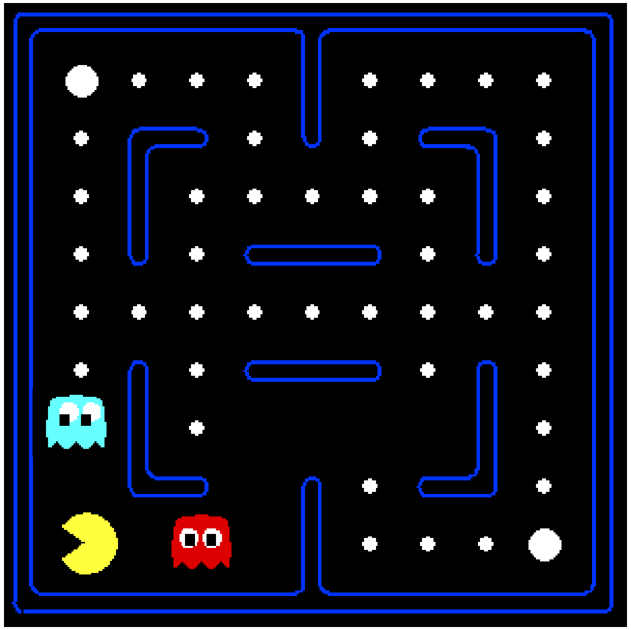


Video of Demo Q-Learning Pacman – Tricky – Watch All



Feature-Based Representations

- Solution: describe a state using a **vector of features** (aka "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

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

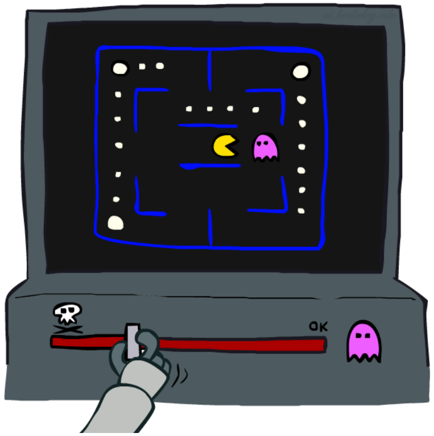
- Q-learning with linear Q-functions:

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

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

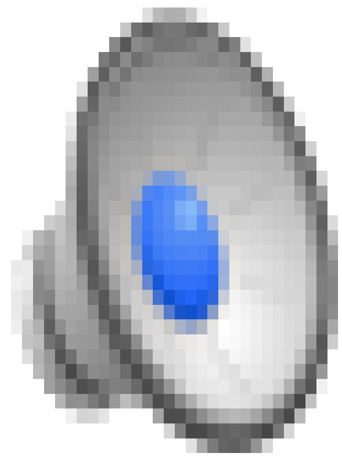
$$Q(s,a) \leftarrow Q(s,a) + \alpha\,[\text{difference}] \qquad \text{Exact Q's}$$

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

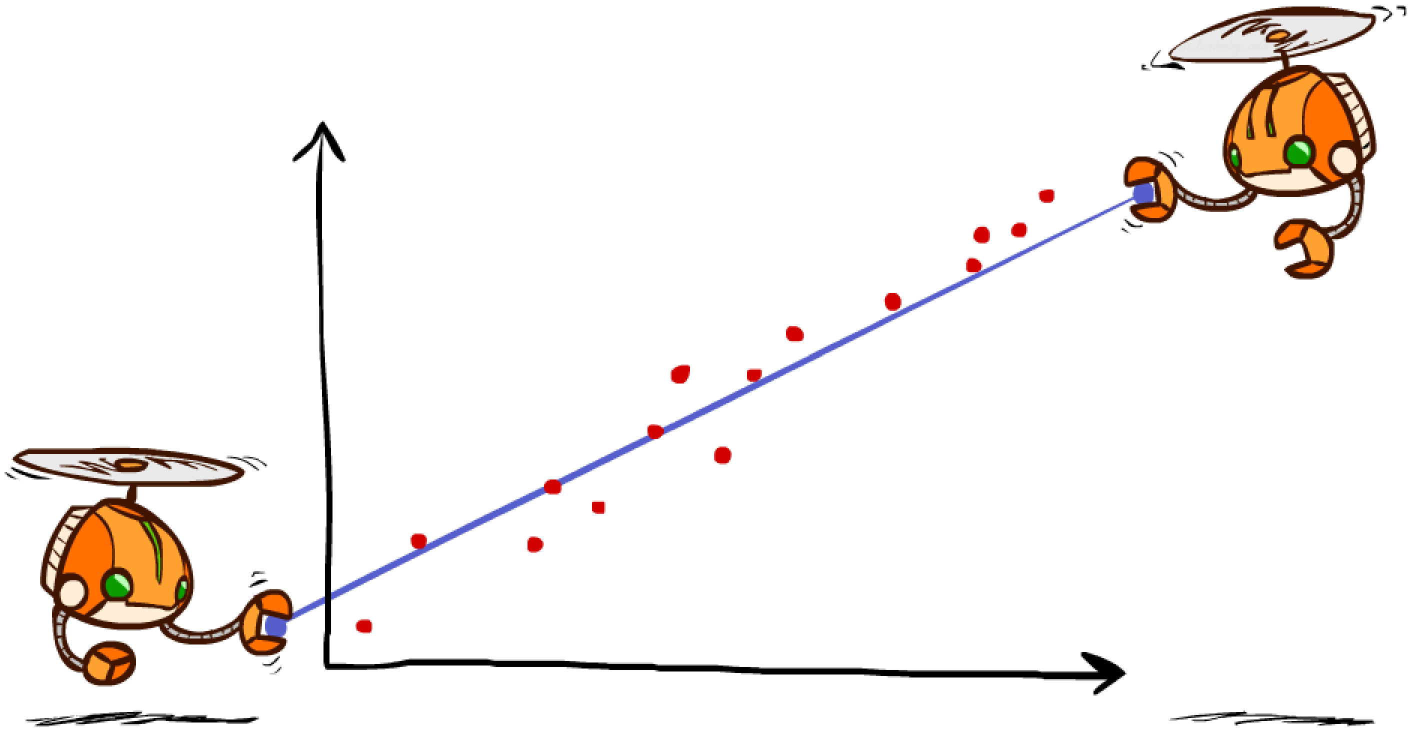


- Intuitive interpretation:
  - Adjust weights of active features
  - E.g., if something unexpectedly bad happens, blame the features that were on: disprefer all states with that state's features
- Formal justification: online least squares

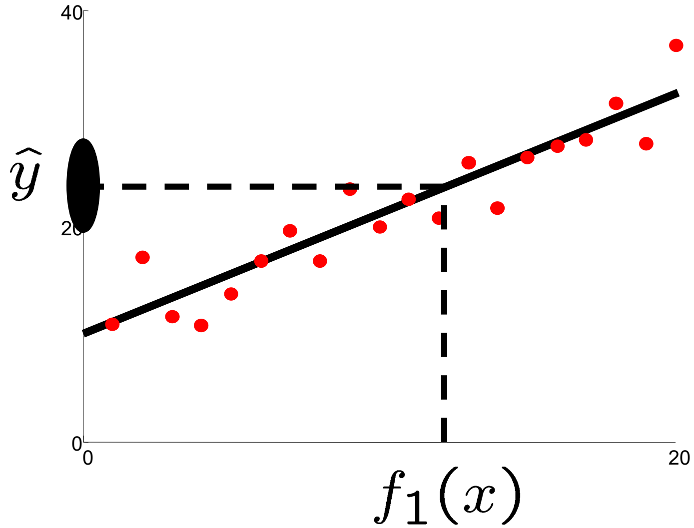
Video of Demo Approximate Q-Learning -- Pacman

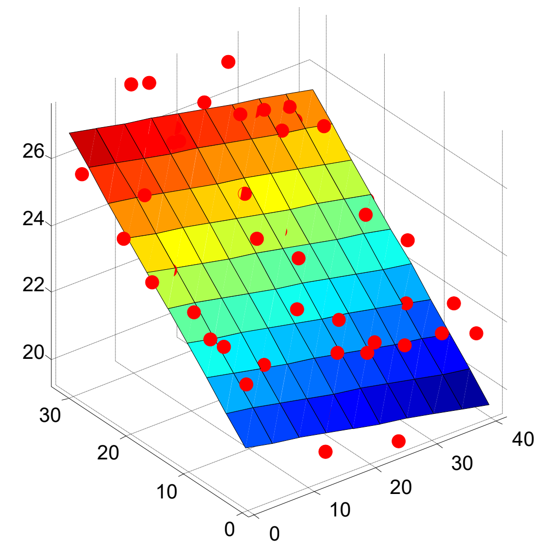


Q-Learning and Least Squares



Linear Approximation: Regression*





Prediction:

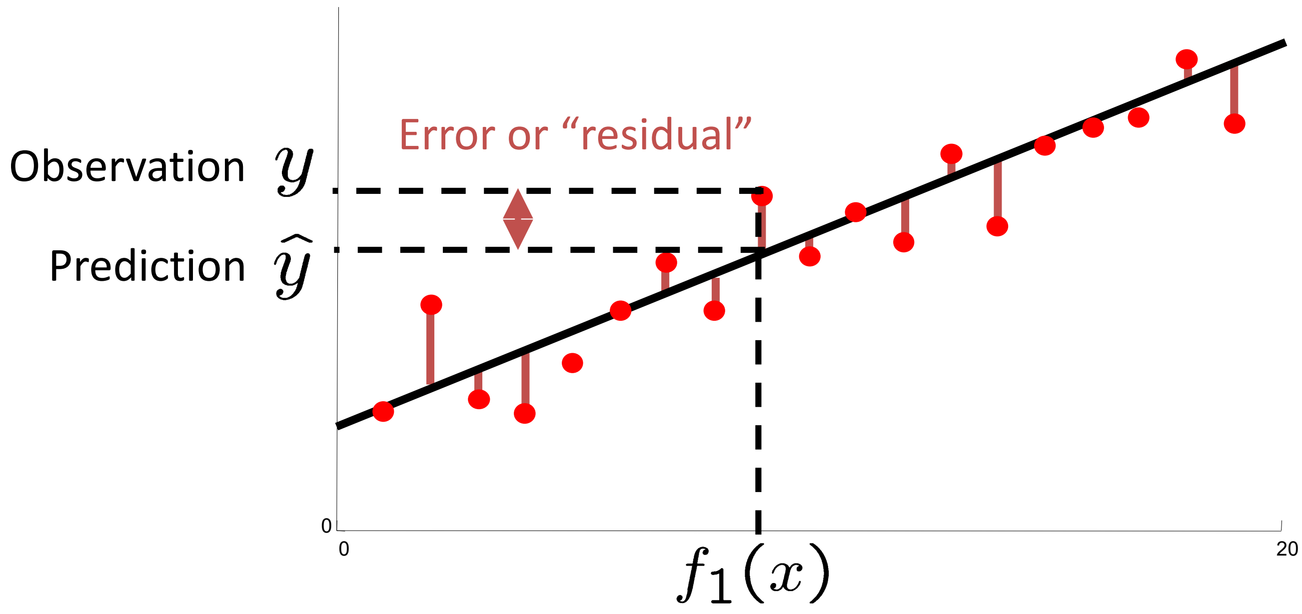
$$\hat{y} = w_0 + w_1 f_1(x)$$

Prediction:

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

Optimization: Least Squares*

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



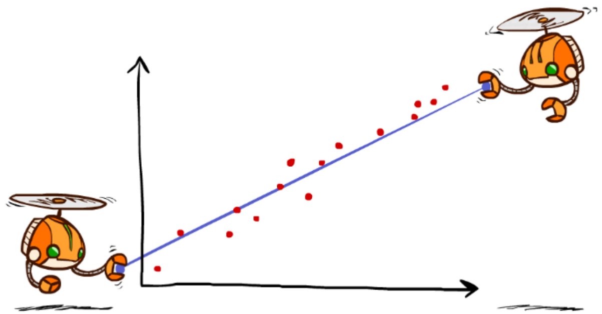
Minimizing Error*

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

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

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

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

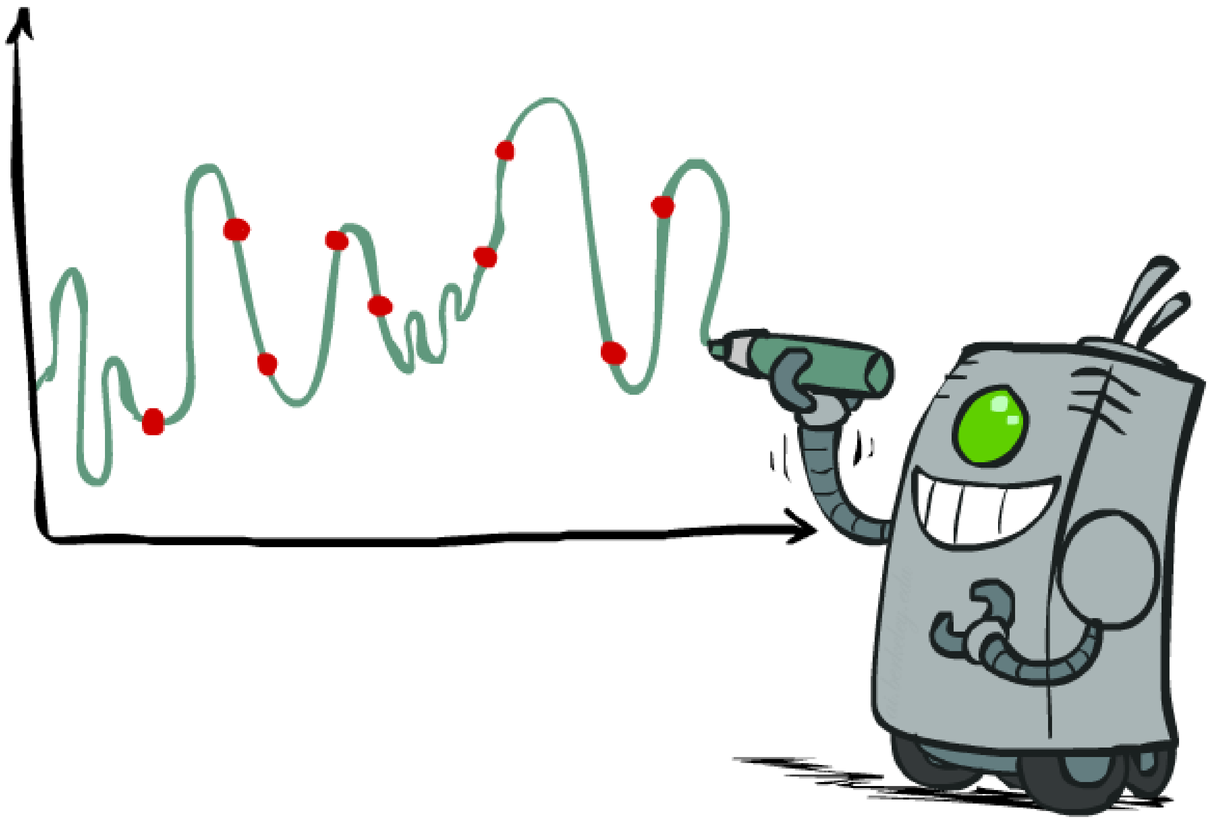


Approximate q update explained:

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

"target" "prediction"

Overfitting: Why Limiting Capacity Can Help*



Example: Inverse Reinforcement Learning

[Video from https://www.youtube.com/watch?v=W_gxLKSsSIE]

Policy Search

[Andrew Ng]

[Video: HELICOPTER]

Applications of RL

- Games
  - Backgammon, Solitaire, Real-time strategy games
- Elevator Scheduling
- Stock investment decisions
- Chemotherapy treatment decisions
- Robotics
  - Navigation, Robocup
  - http://www.youtube.com/watch?v=CIF2SBVY-J0
  - http://www.youtube.com/watch?v=5FGVgMsiv1s
  - http://www.youtube.com/watch?v=W_gxLKSsSIE
  - https://www.youtube.com/watch?v=_Mmc3i7jZ2c
- Helicopter maneuvering