CS 6511: Artificial Intelligence

Reinforcement Learning

Amrinder Arora

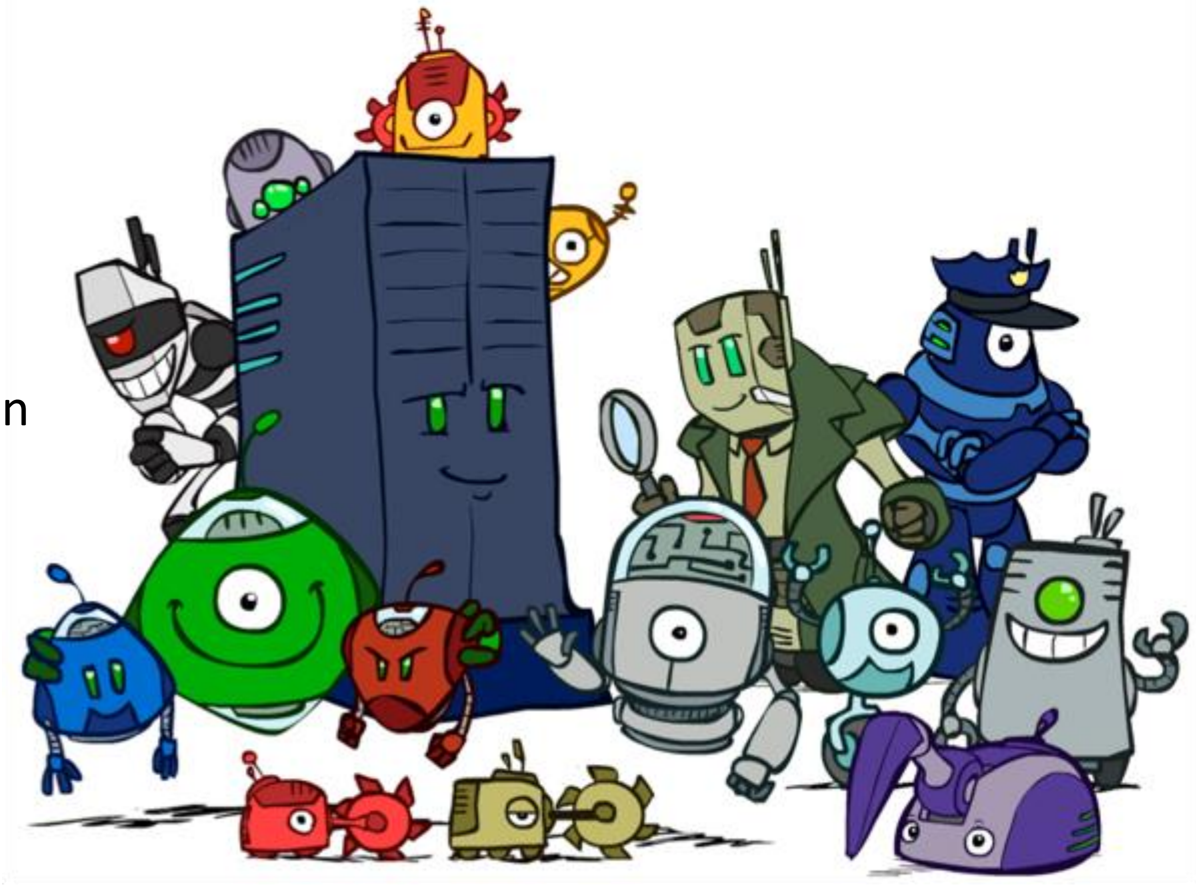
The George Washington University

[Original version of these slides was created by Dan Klein and Pieter Abbeel for Intro to AI at UC Berkeley. http://ai.berkeley.edu]

Where we are…

- We've seen how AI methods can solve problems in:
  - Searching for a state
  - Searching for a solution (CSP)
  - Searching for victory (Games)
  - Decision Making in Uncertainty (Markov Decision Problems)

- Next up: More Uncertainty and Learning!
  - Making decisions when we don't know the environment dynamics
  - Learning about the environment

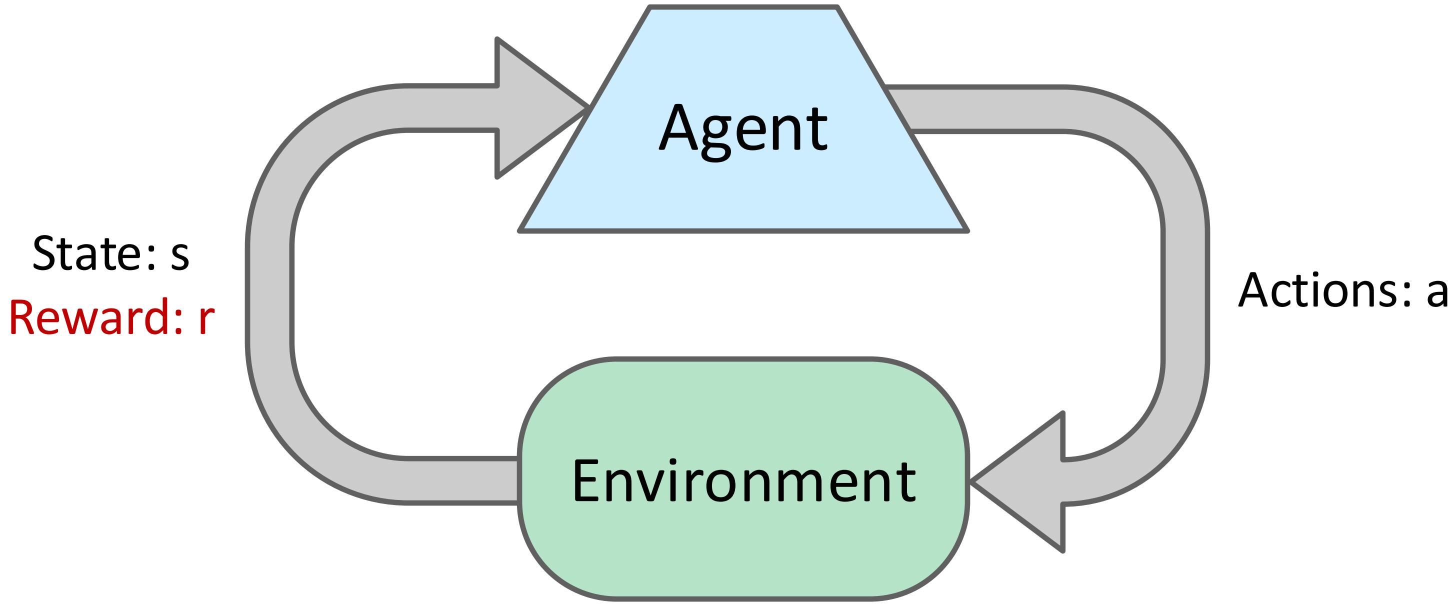


3 Forms of Learning

- Supervised Learning: Given training data set, learn the pattern
- Unsupervised Learning: No training data set given, just learn/observe
- Reinforcement Learning: No training data set given, but you can make an action and get some feedback. This feedback can be considered a "mini training episode", with real cost (or reward).

Not to be confused with habituation, sensitization and associative learning – 3 Mechanisms by which people, or dogs, or sea slugs learn! Refer to the "Learning to Learn" lecture.

Reinforcement Learning



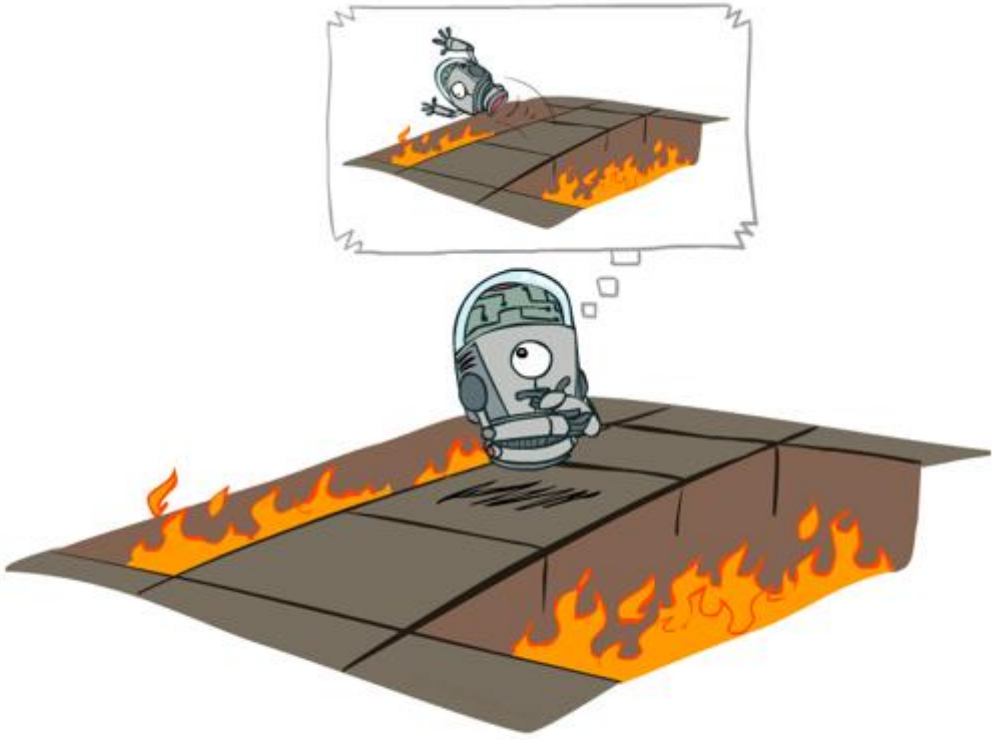
- Basic idea:
  - Receive feedback in the form of rewards
  - Agent's utility is defined by the reward function
  - Must (learn to) act so as to maximize expected rewards
  - All learning is based on observed samples of outcomes!

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 $T(s,a,s')$
  - A reward function $R(s,a,s')$
- Still looking for a policy $\pi(s)$

- New twist: don't know T or R
  - That is, 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)

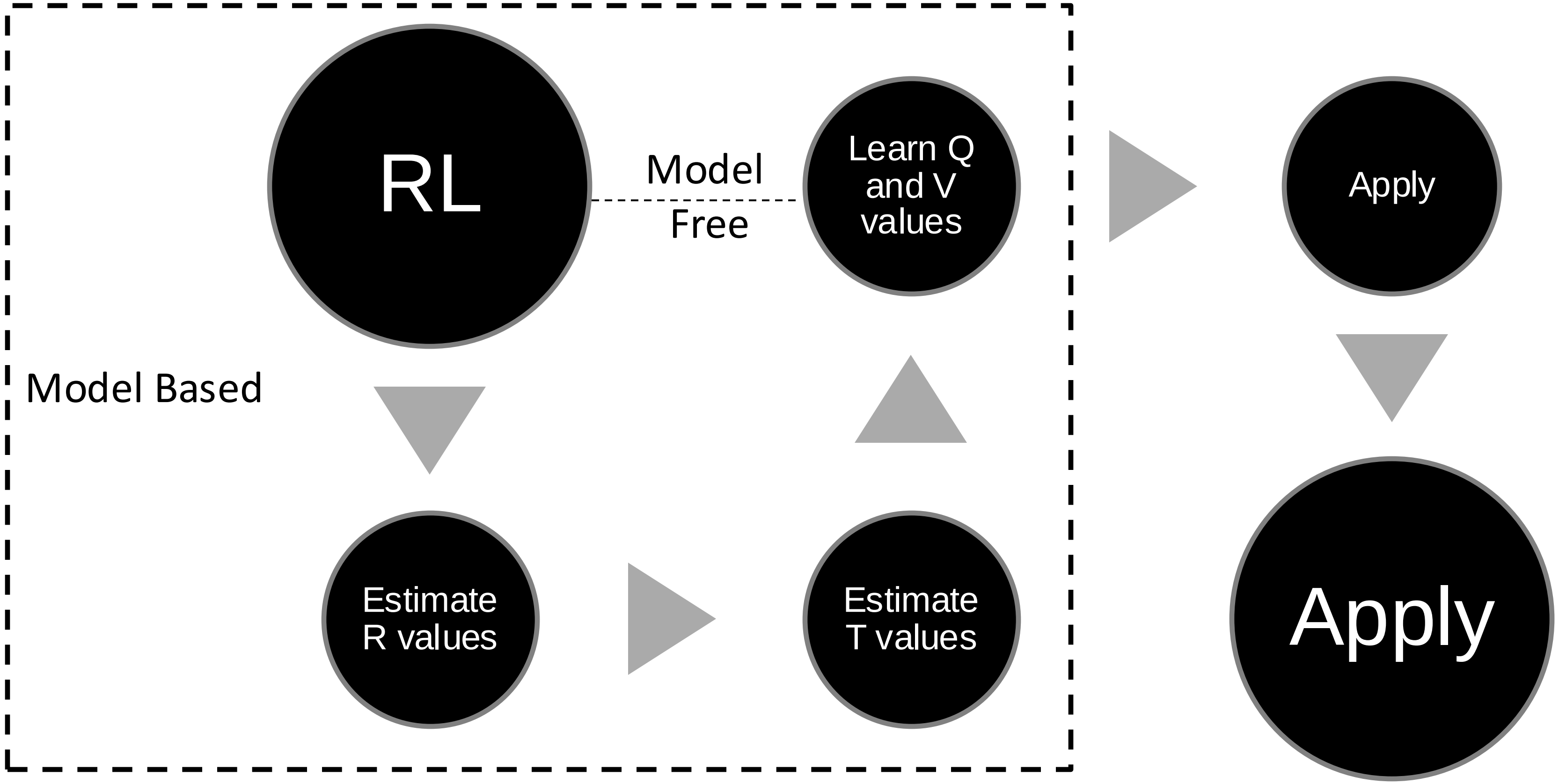


Offline Solution

Online Learning

Two Broad Categories

- Model Based – We will learn the MDP model (T, R, …)
- Model Free – We learn the Q, V values directly

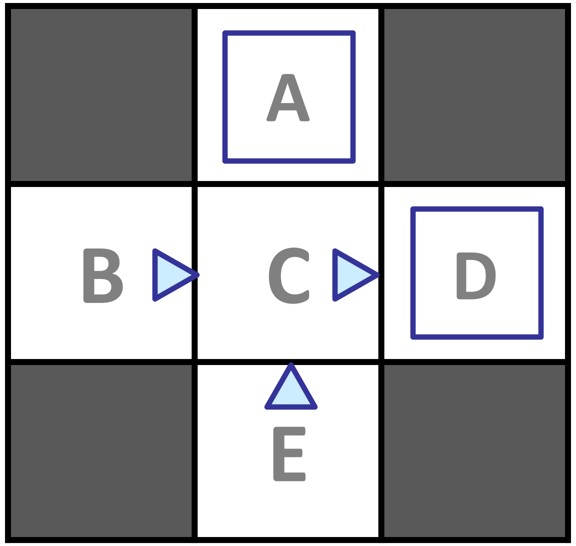


Model-Based Learning

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

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
…

Model-Free Learning

- A key mechanism to learn in MDP settings
- In this, we don't try to learn T and R values. We learn Q and V values directly.

- Subtopics
  - Passive RL – Evaluating a policy V/Q values for given policy
  - Active RL – Learn the policy also
  - Q-Learning – Learn the Q values, using Exponential Moving Average (EMA)
  - EMA - Approach

Exponential Moving Average

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

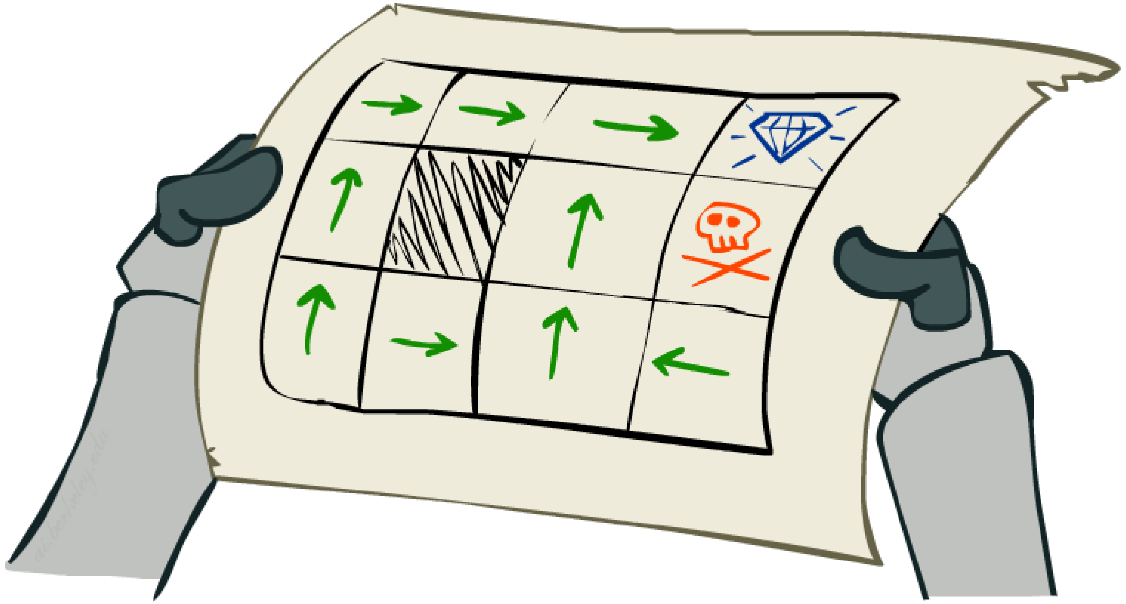
$$\bar{x}_n = \frac{x_n + (1-\alpha)\cdot x_{n-1} + (1-\alpha)^2\cdot x_{n-2} + \dots}{1 + (1-\alpha) + (1-\alpha)^2 + \dots}$$

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

- Decreasing learning rate (alpha) can give converging averages

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 Evaluation

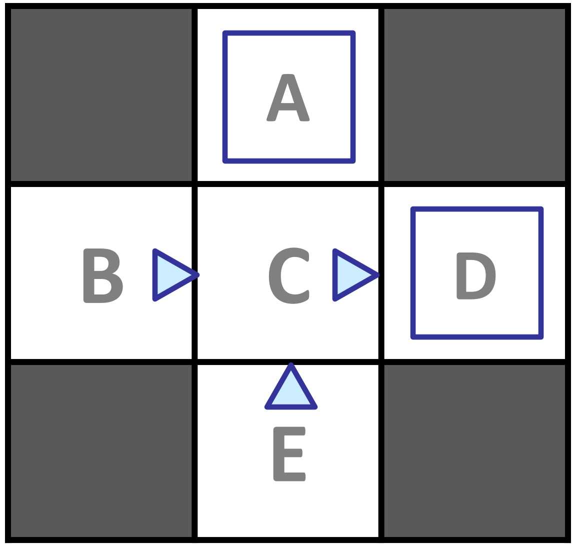
- Goal: Compute values for each state under π

- 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

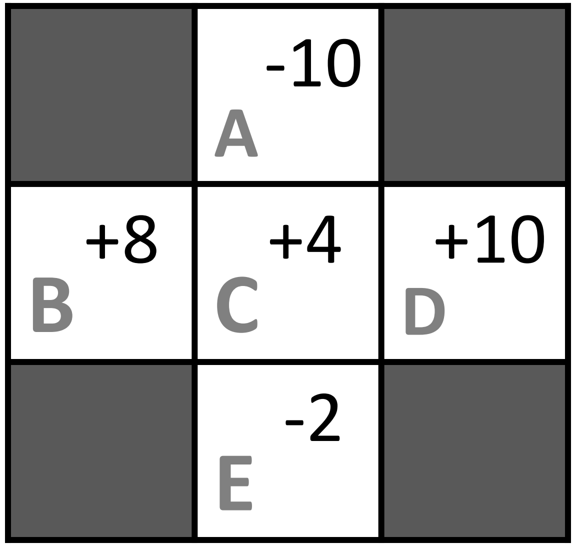
Episode 3

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

Episode 4

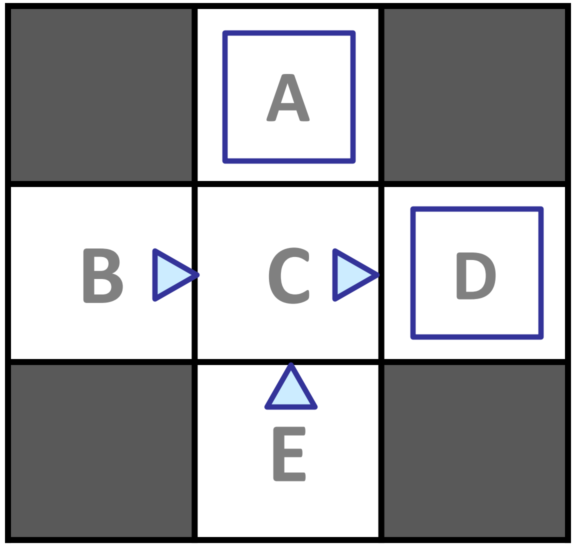
E, north, C, -1
C, east, A, -1
A, exit, x, -10

Output Values



Example: Direct Evaluation

Input Policy π



Assume: $\gamma = 1$

B = 0.8 * C + 0.1 * B + 0.1 * B

➔ B = C

E = 0.8 * C + 0.1 * E + 0.1 * E. E = C

Q(C,(E)) = 0.8 * D + 0.1 * A + 0.1 * E
C = 8 -1 + 0.1 * C
C * 0.9 = 7

Q(C,(S)) = 0.8 * E + 0.1 * B + 0.1 * D
C = 0.9 C + 1

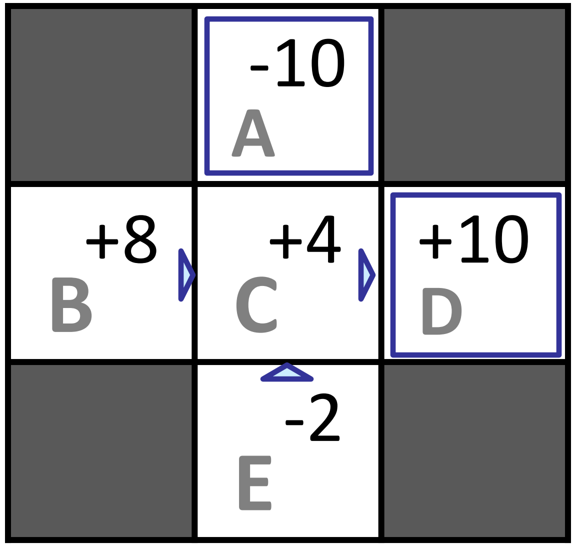
0.1 C = 1
C = 10

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



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

Sample-Based Policy Evaluation?

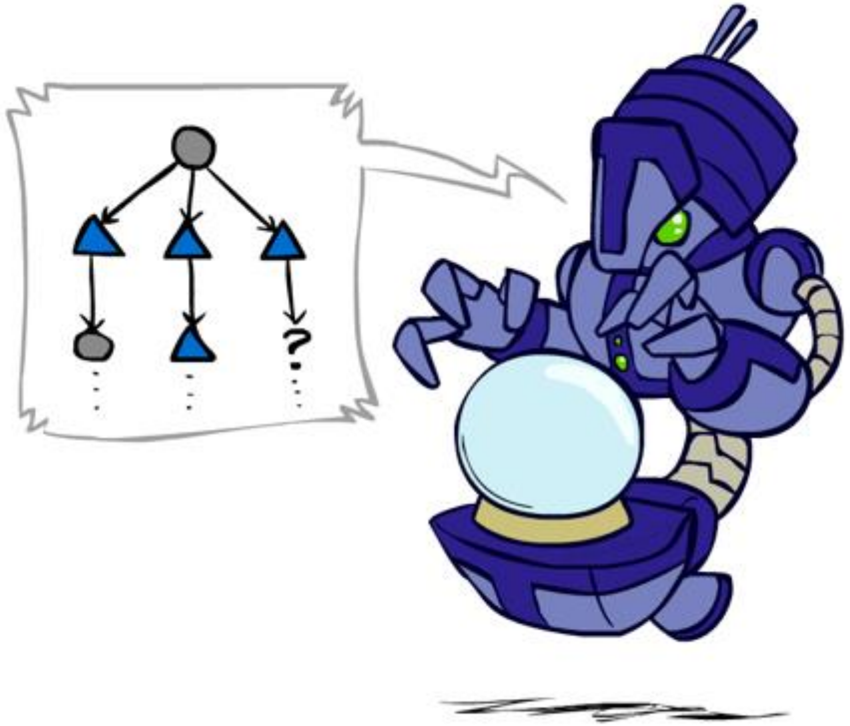
- We want to improve our estimate of V by computing these averages:

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

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

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

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



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…

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:

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

In MDPs in general

$$V^*(s) = \max_a Q^*(s,a)$$

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

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

- From Q values, we can compute V values trivially
- From V values, we can compute Q values, but that takes some computation..
- Therefore, if you only want to compute and store one set of values, Q values is an easier choice.

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,s') with a living reward of r
  - 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

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

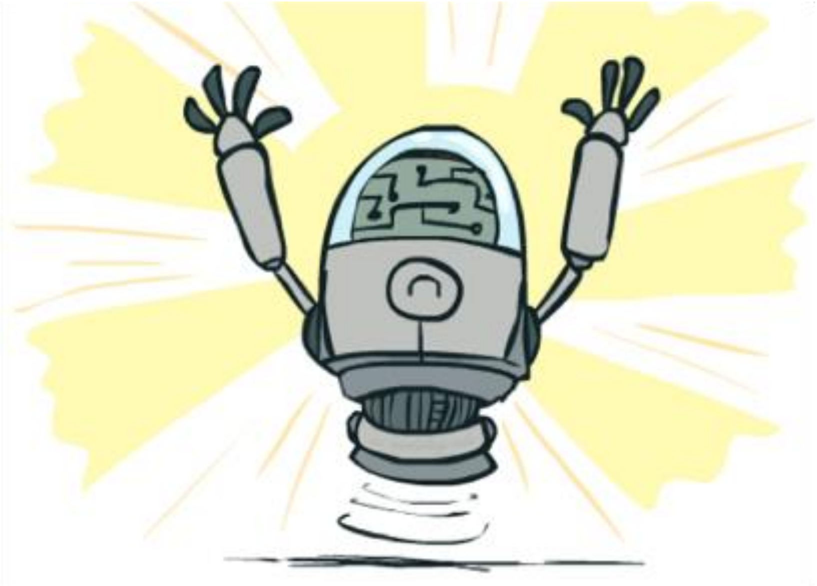
Q-Learning

- UpdateQValues(Q)
- In state s, choose action a → Env returns s',r
- Use Max to compute vs' -> This uses current Q
- Q(s,a) = (1-alpha) Q(s,a) + alpha * (r + gamma * vs')

Q-Learning Properties

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

- This is called off-policy learning



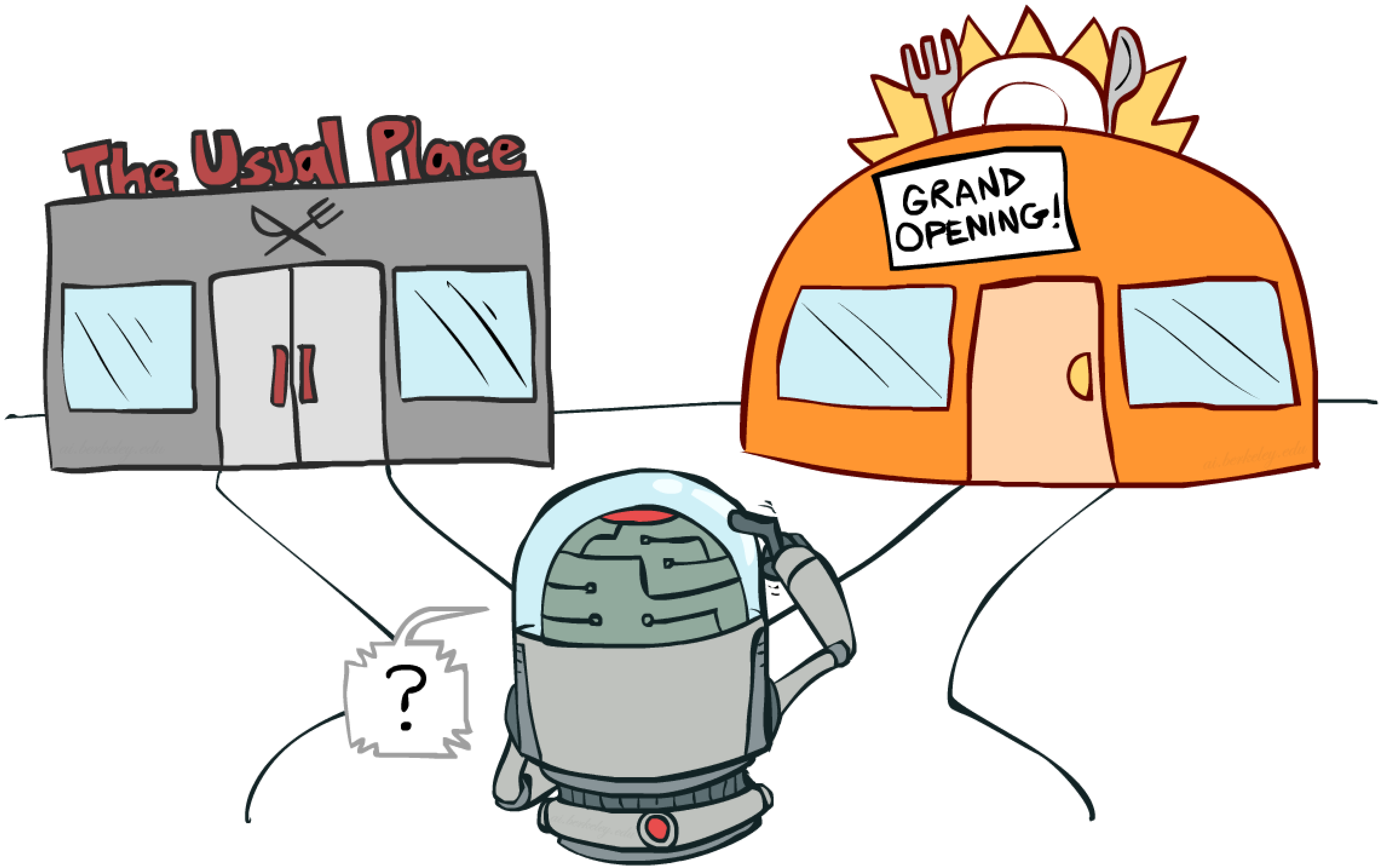
- Caveats:
  - You have to explore enough
  - You have to eventually make the learning rate small enough
  - … but not decrease it too quickly
  - Basically, in the limit, it doesn't matter how you select actions (!)

Time to Recap!

- RL is simple.
- RL is easy.
- RL is intuitive.
- RL (online learning) is very different from MDPs (offline planning)
- We can explain what RL is, in a few sentences to someone who doesn't know RL, including two broad categories of RL.

(Try this in the breakout room.)

Exploration vs. Exploitation



Schemes for Forcing Exploration

Random

Exploration Function (Count/density based)

Regret (Result based)

Random Exploration

- Simplest: random actions (ε-greedy)
  - Every time step, flip a coin
  - With (small) probability ε, act randomly
  - With (large) probability $1-\varepsilon$, 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

Exploration Functions

- Main idea
  - Explore areas whose badness is not (yet) established, eventually stop exploring

- How to implement
  - 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: $$Q(s,a) \leftarrow_\alpha R(s,a,s') + \gamma \max_{a'} Q(s',a')$$

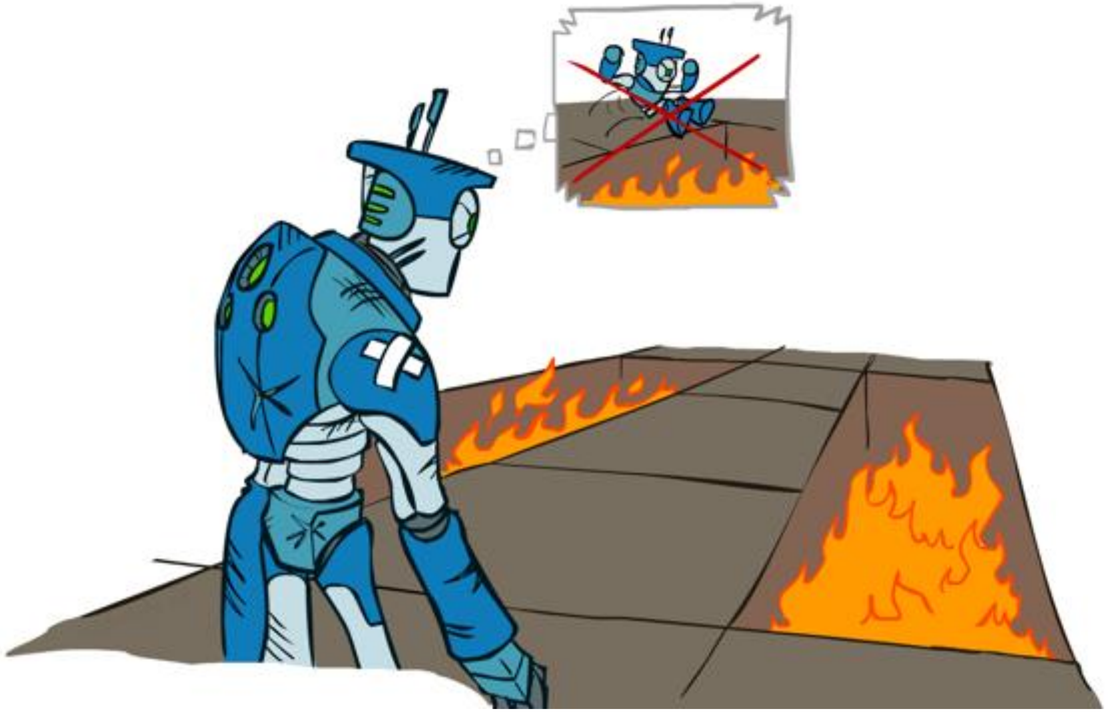
Modified Q-Update: $$Q(s,a) \leftarrow_\alpha R(s,a,s') + \gamma \max_{a'} f(Q(s',a'),N(s',a'))$$

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

Regret

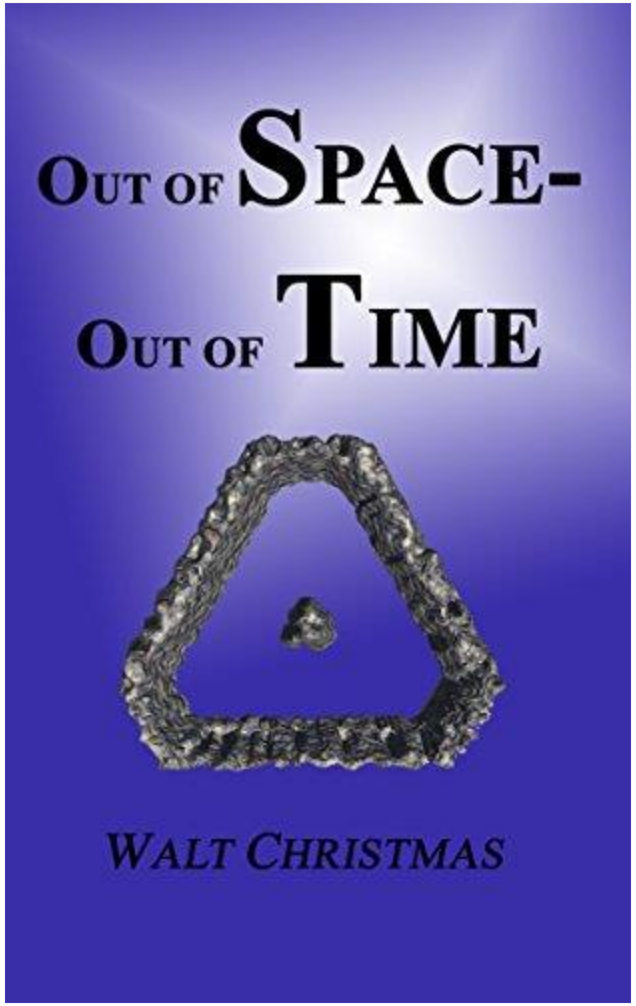
- Even if you learn the optimal policy, you still make mistakes along the way!
- Regret is a measure of your total mistake cost: the difference between actual rewards and optimal (expected) rewards

Empirically: Random exploration and exploration functions both end up optimal, but random exploration has higher regret.



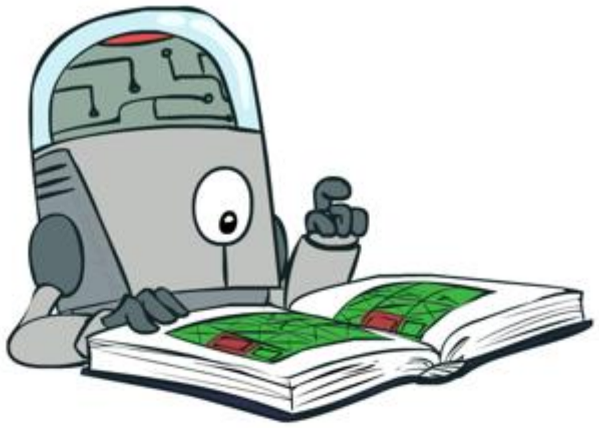
Despite all our efforts…

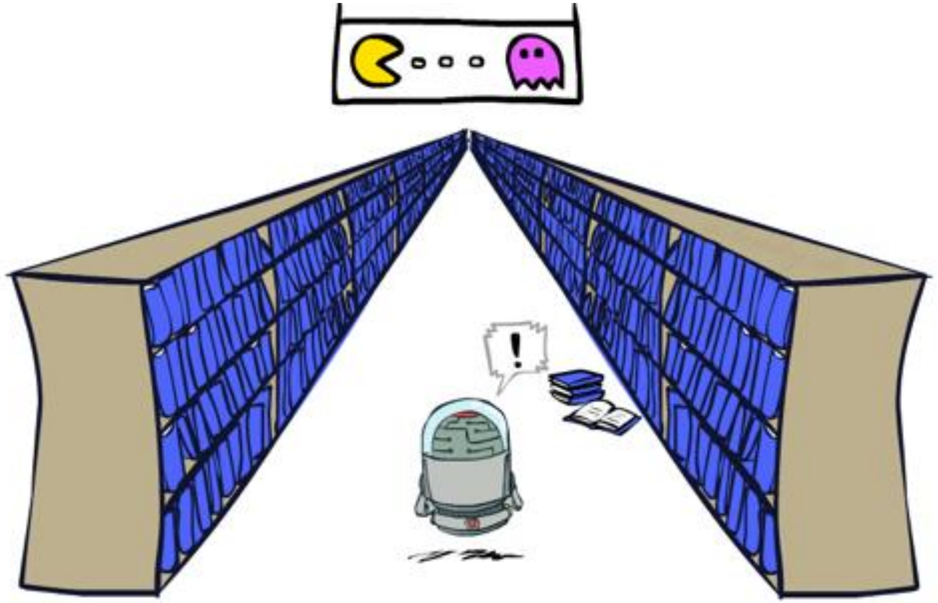
REINFORCEMENT LEARNING IS OUT OF SPACE



Generalizing Across States

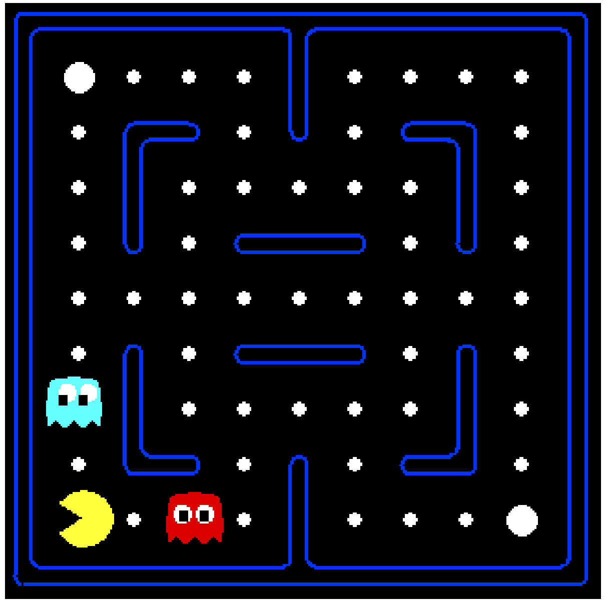
- Basic Q-Learning keeps a table of all q-values
- In realistic situations, we cannot possibly learn about every single state!
  - Too many states to visit them all in training
  - Too many states to hold the q-tables in memory
- Instead, we want to generalize:
  - Learn about some small number of training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



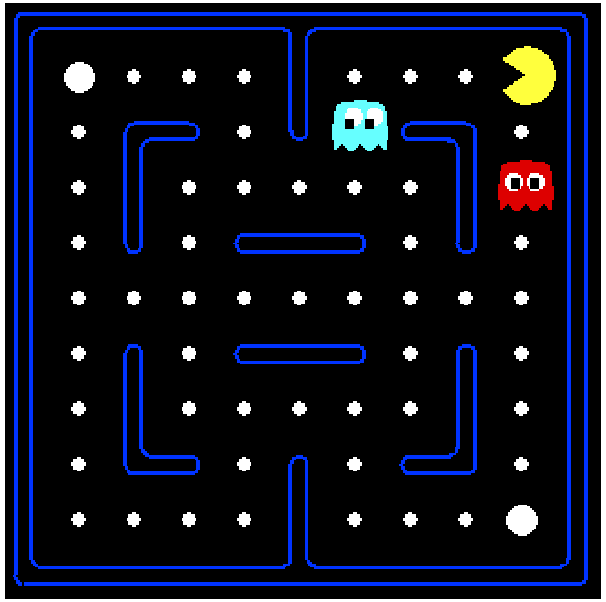


Example: Pacman

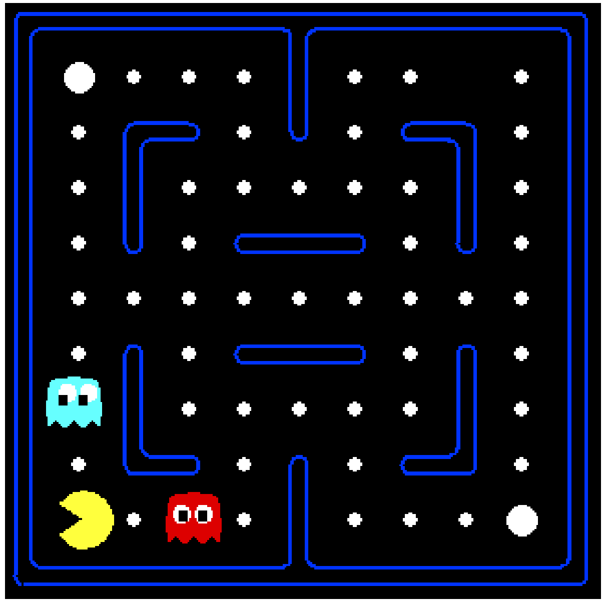
Let's say we discover through experience that this state is bad:



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

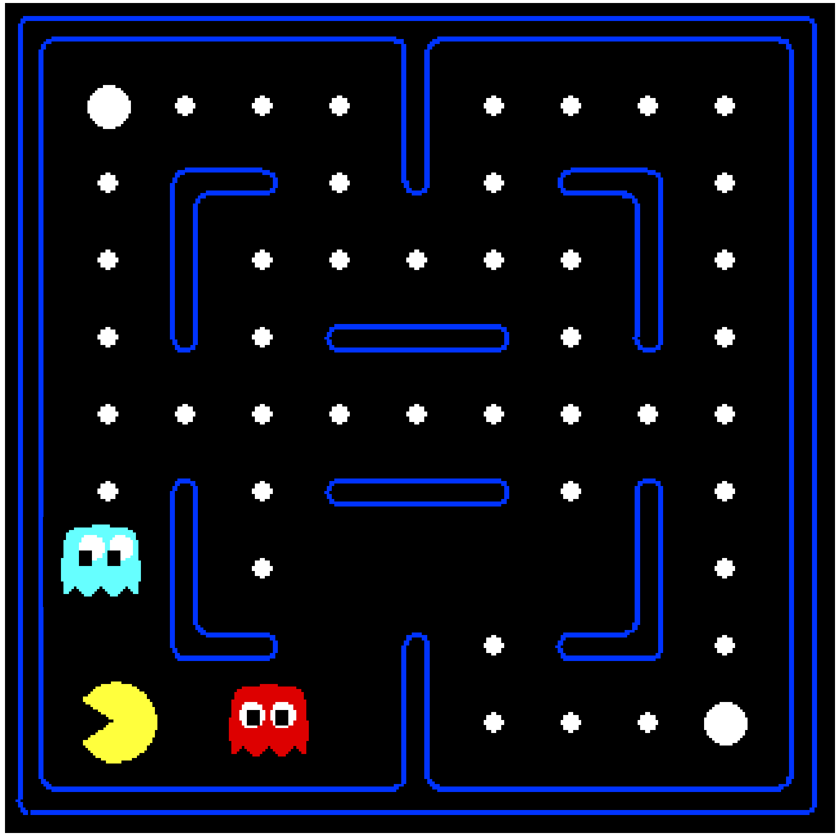


Or even this one!



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

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

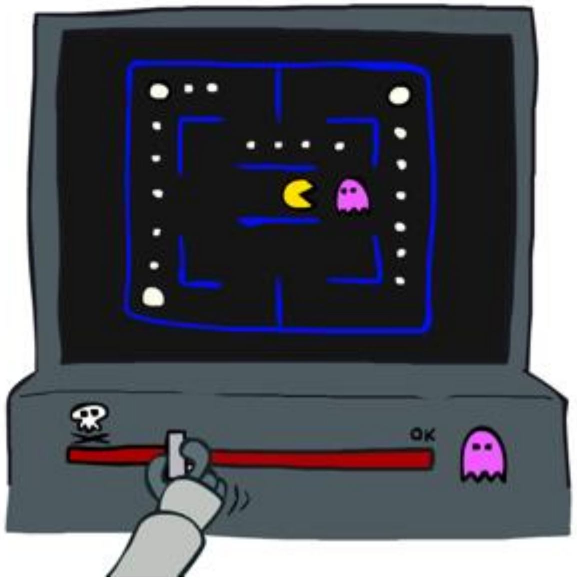
- Q-learning with linear Q-functions:

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

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

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

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

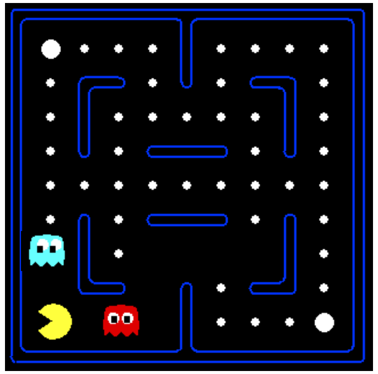


- Intuitive interpretation:
  - Adjust weights of active features
  - E.g., if something unexpectedly bad happens, blame the features that were on: disprefer all states with that state's features
- Formal justification: online least squares

Example: Q-Pacman

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

s



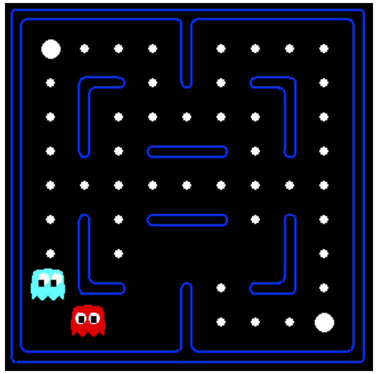
$$f_{DOT}(s, \text{NORTH}) = 0.5$$

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

$a = \text{NORTH}$

$r = -500$

s'



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

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

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

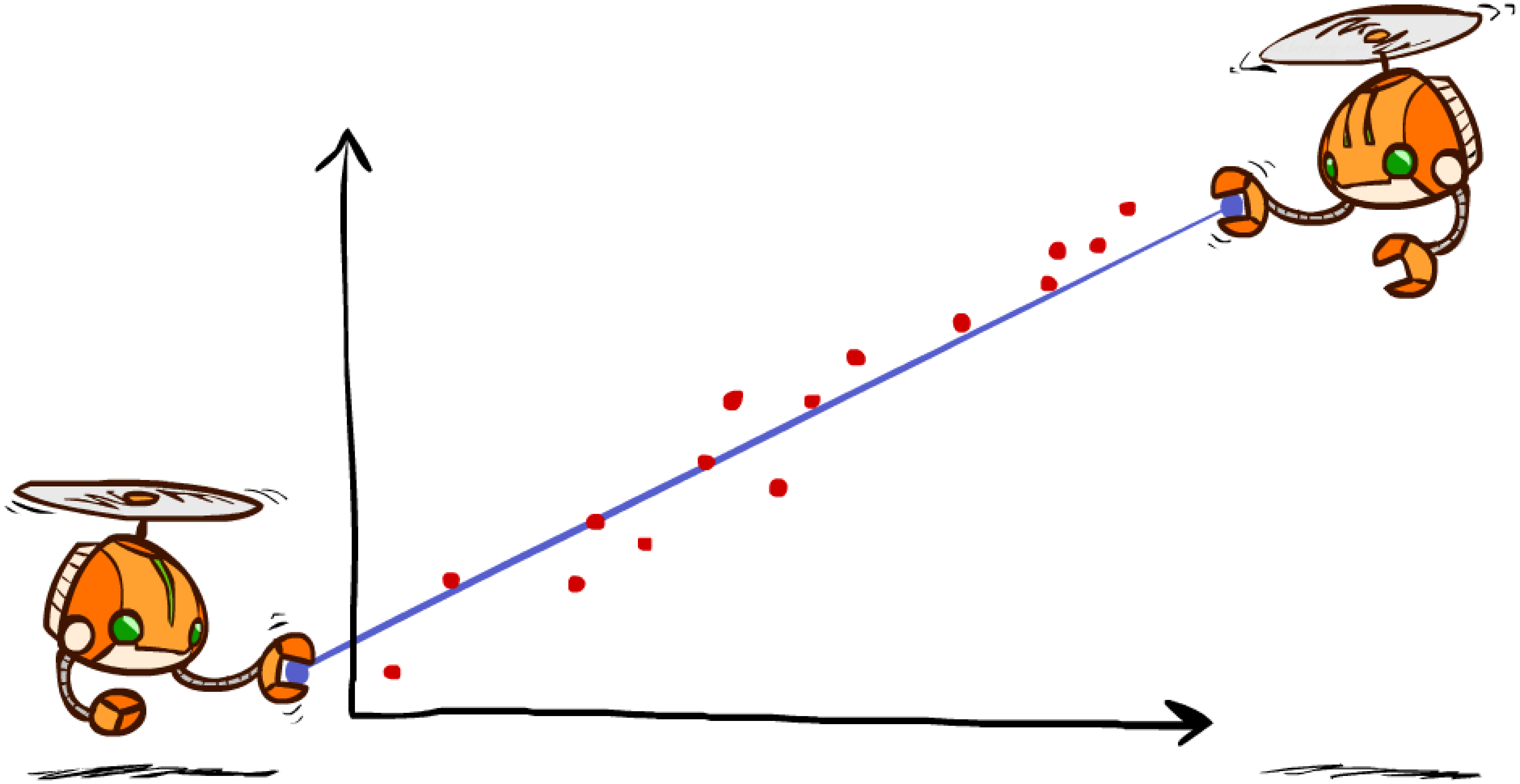
$$\text{difference} = -501$$

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

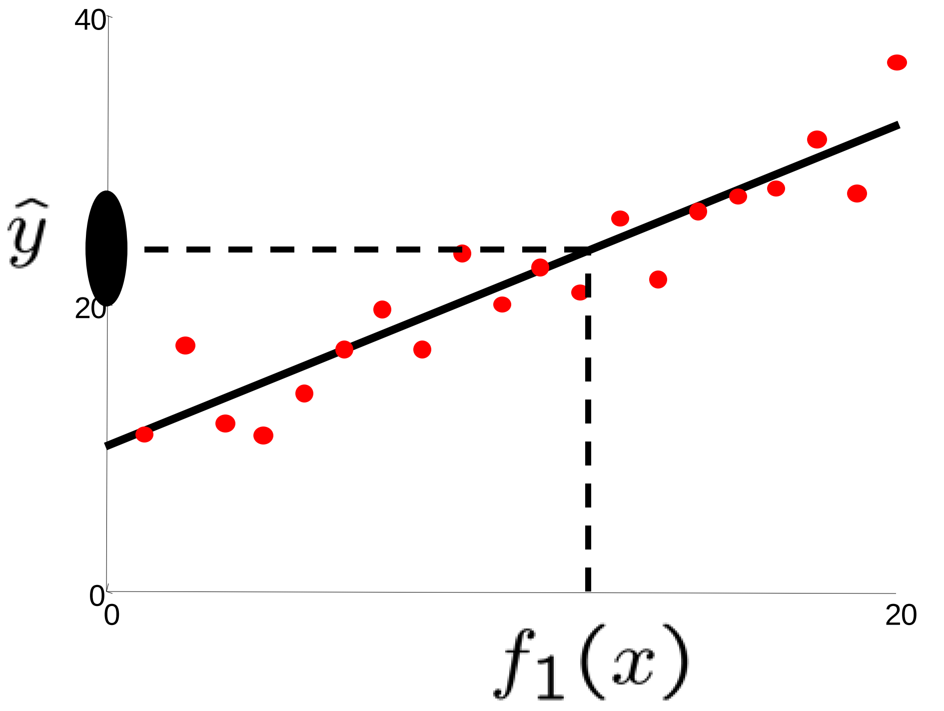
$$w_{GST} \leftarrow -1.0 + \alpha\,[-501]\,1.0$$

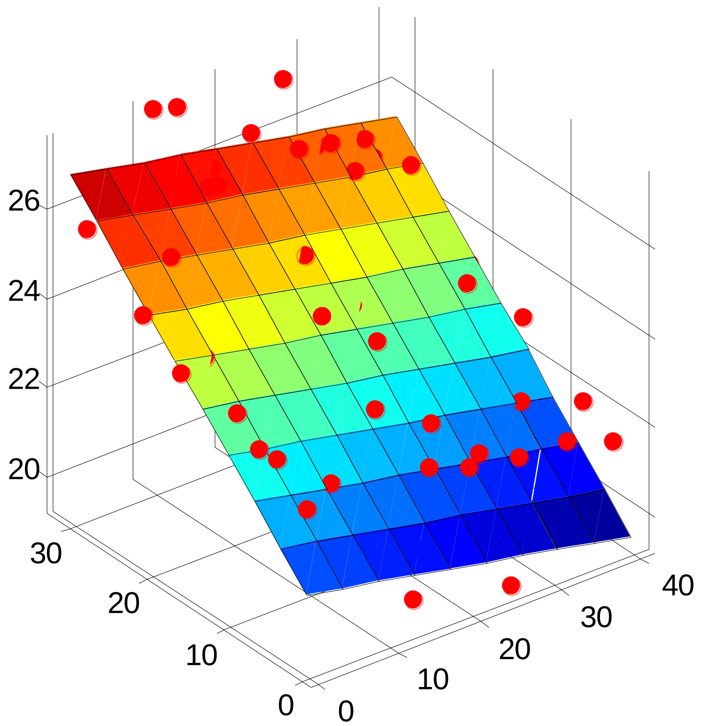
$$Q(s,a) = 3.0 f_{DOT}(s,a) - 3.0 f_{GST}(s,a)$$

Q-Learning and Least Squares



Linear Approximation: Regression*





Prediction:

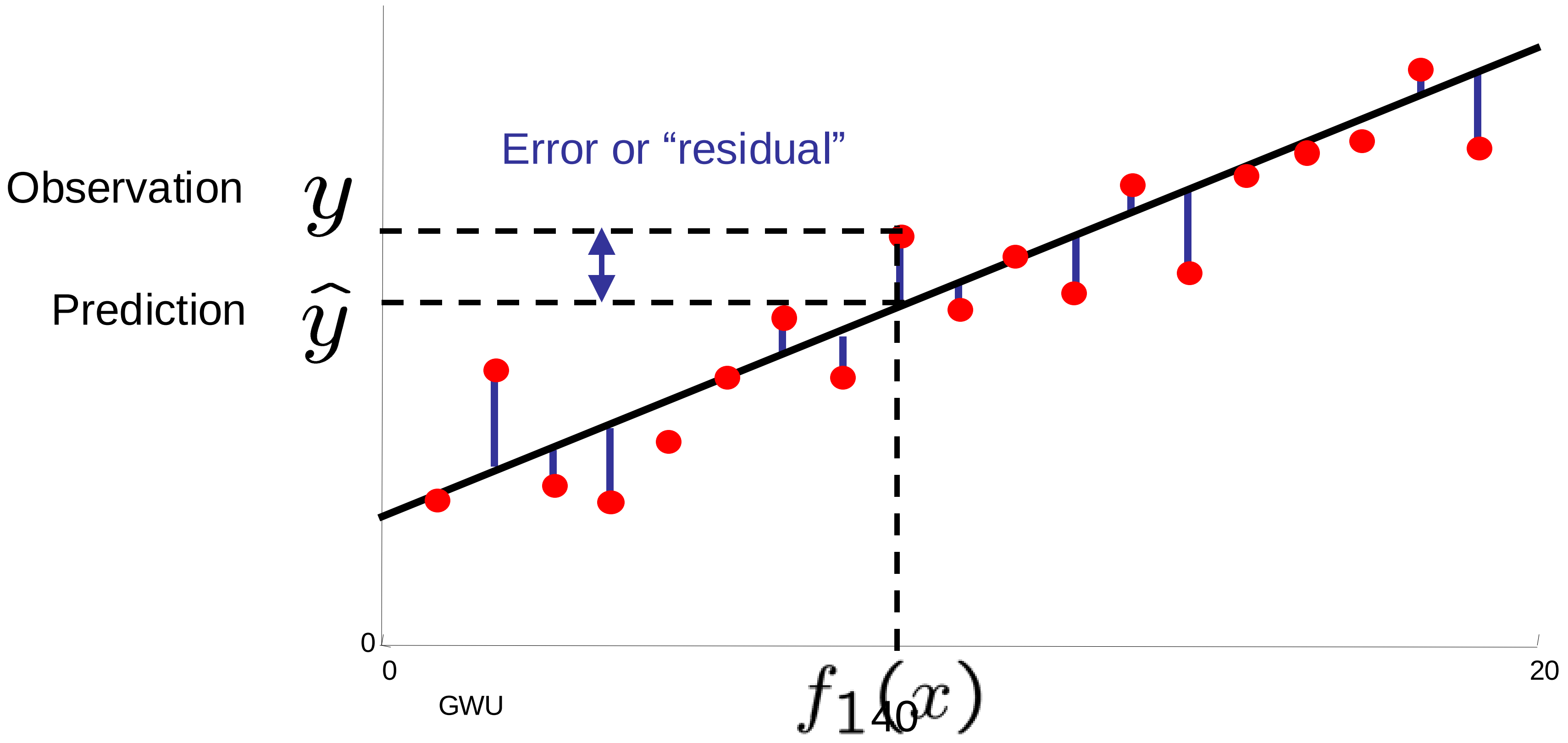
$$\hat{y} = w_0 + w_1 f_1(x)$$

Prediction:

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

Optimization: Least Squares*

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



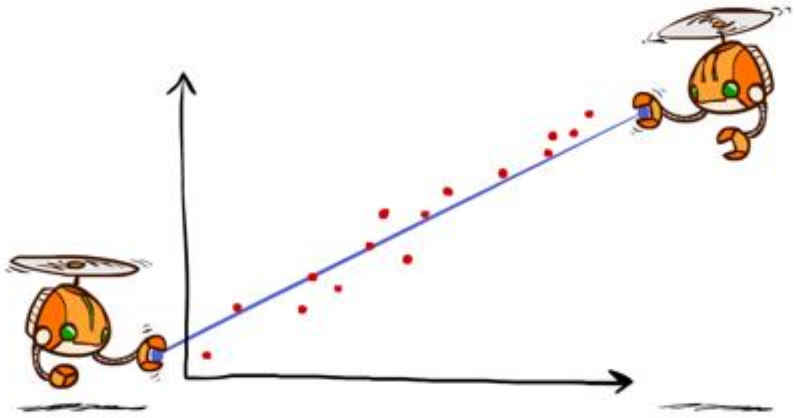
Minimizing Error*

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"

Credit Assignment Problem

- Not easy to identify credit for each move in a Chess game
- If credit is only given at the end of the game, then..
  - Many good moves can get a negative credit if the end result is a loss
  - Many bad moves can get a positive credit if the end result is a win
  - Many many games need to be played before learning really happens
- One solution is to give rewards early on (Reward Shaping)
- If we try to give rewards early on, then..
  - Agent will maximize on those rewards, not the actual outcome

Reinforcement Learning Application Examples

- https://sagemaker-examples.readthedocs.io/en/latest/reinforcement_learning/rl_mountain_car_coach_gymEnv/rl_mountain_car_coach_gymEnv.html

Let’s (not) get historical

- Q-learning was introduced by Chris Watkins in 1989.
- Convergence proof by Watkins and Dayan in 1992.
- In 1981 “Delayed reinforcement learning”, presented by Bozinovski's Crossbar Adaptive Array (CAA).
- The term “secondary reinforcement” is borrowed from animal learning theory, to model state values via backpropagation: the state value of the consequence situation is backpropagated to the previously encountered situations.
- In 2014, Google DeepMind patented an application of Q-learning to deep learning, titled "deep reinforcement learning" or "deep Q-learning" that can play Atari 2600 games at expert human levels.
- https://patentimages.storage.googleapis.com/71/91/4a/c5cf4ffa56f705/US20150100530A1.pdf

Summary

- Introduction
  - What is Reinforcement Learning
  - Handling MDPs, when we don't know T and R functions.
- Two broad categories of Reinforcement Learning (RL)
  - Model Based - Simply try and learn T and R values. Then, calculate Q, V as usual.
  - Model Free - Don't worry about T and R values. Learn Q, V values directly.
    - Q-Learning: Algorithm to learn Q values by trying. Update Q value using something like exponential moving average
    - [A useful background technique - Exponential Moving Average]
- Exploration vs. exploitation in RL
  - Quantify exploration vs. exploitation
  - 3 methods: Random, Exploration function, Regret
  - How much exploration to do - how to make it "time" based (Like in case of simulated annealing)
  - How to make it time based for each state, action combination (Exploration can go down with time)
- Advanced Topics
  - What is credit assignment problem in RL?
  - Is it more of a problem in case of episodic environment or non-episodic environments?
  - How we can use reward shaping (and what are the problems associated with it)?
  - [Not discussed in class] How can we make a generic technique for reward shaping that is not environment based?

10 AI Commandments

1. *"No model is perfect, but some models are useful" General AI and ML*
2. *"The algorithms that forget their history are doomed to repeat it." Graph Search vs. Tree Search*
3. *"Ask not what the state can do for you, ask what you can do in that state." Successor function concept in search problems*
4. *"Your direction is more important than your speed" Informed search vs. uninformed search*
5. *"Fail early. Fail often. Fail forward". Constraint Satisfaction Problems*
6. *"Reality dishes out experiences using probability, not plausibility." Expectimax and MDPs*
7. *"The doer alone learneth." Reinforcement Learning*

Conclusion

- We're done with Part I: Search and Planning!
- We just started Learning!