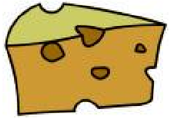
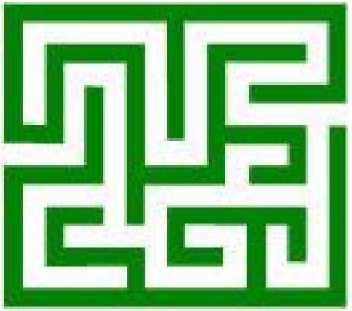
Introduction to Reinforcement Learning

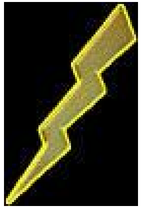
Terran Lane <tdrl@google.com>
Google, Inc.
20 Jul, 2015

Meet Mac the Mouse

?



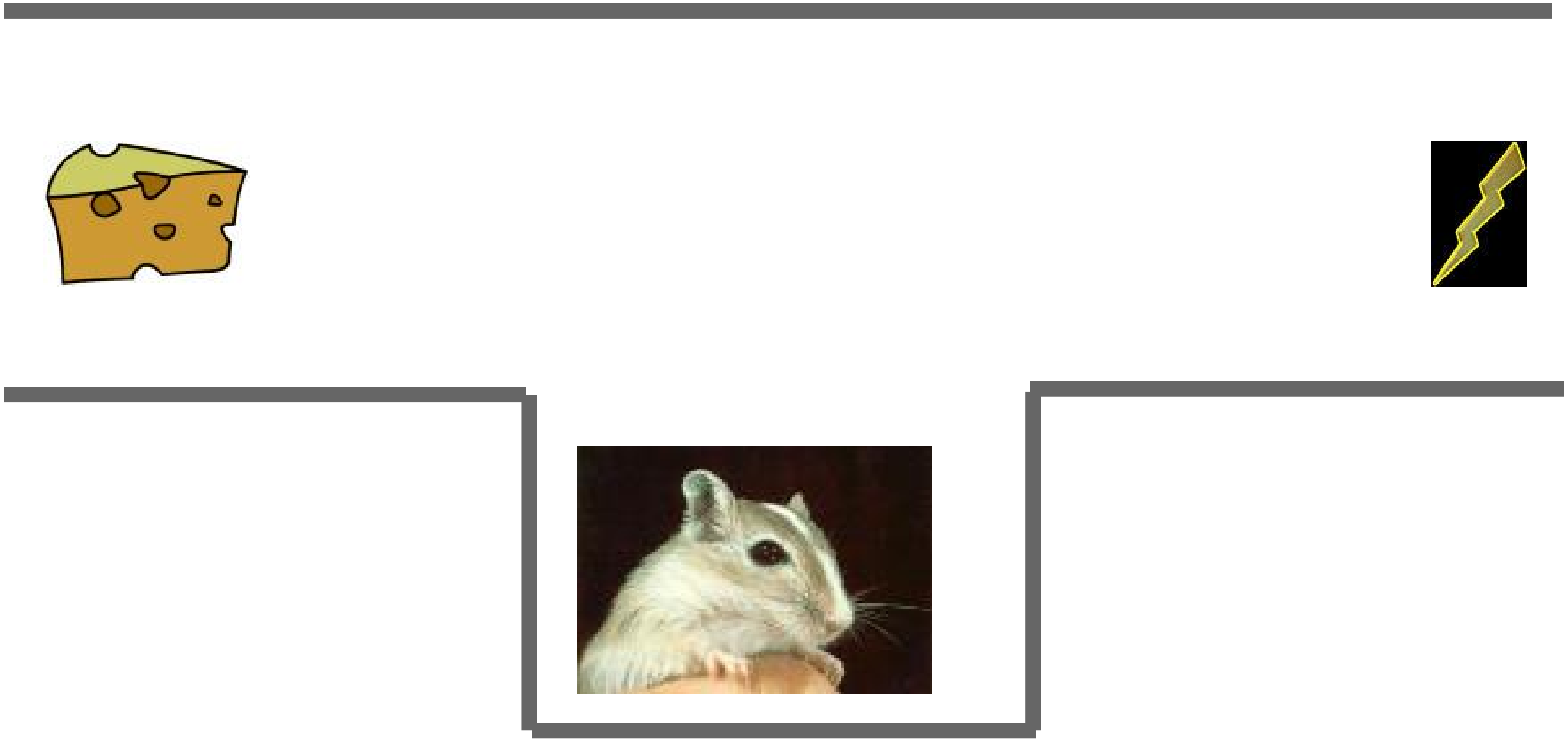




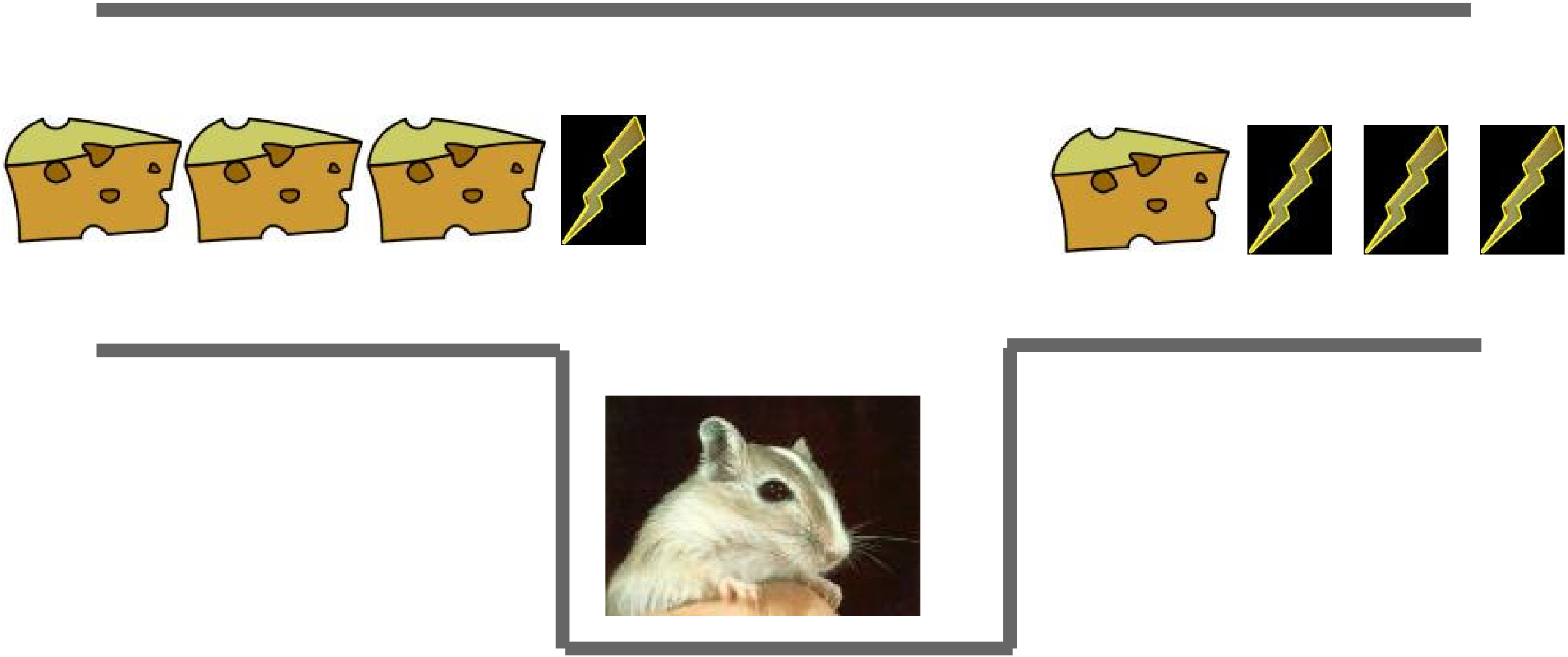
- Mac lives a hard life as a psychology test subject
- Runs around mazes all day, finding food and dodging shocks
- Has to *learn* how to behave: find food, avoid shocks

The Reinforcement Learning Problem

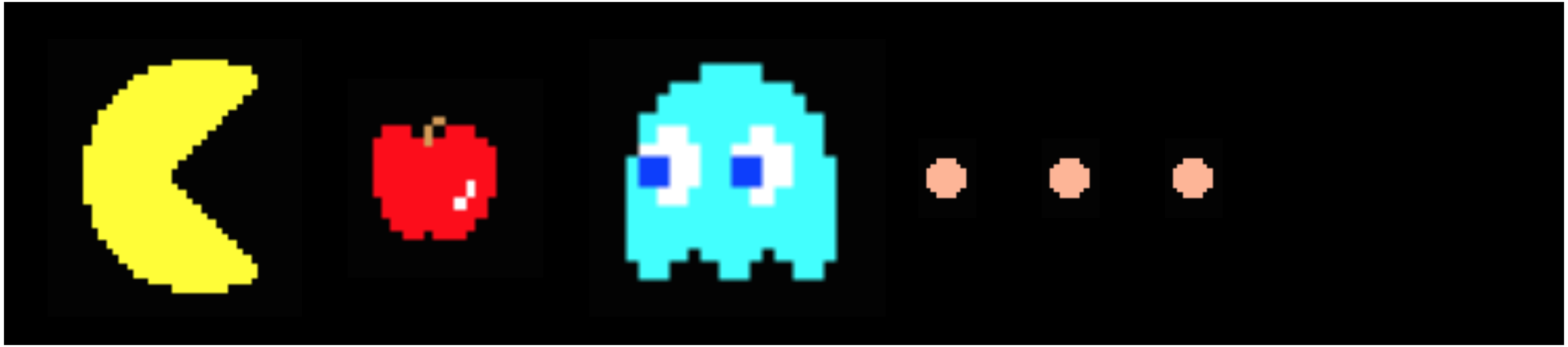
- Learning *control*: how to act to achieve goals (rewards)
- Supervised learning: mapping from feature vector to output value
- RL: mapping from feature vector to action
- Catch: *delayed rewards*



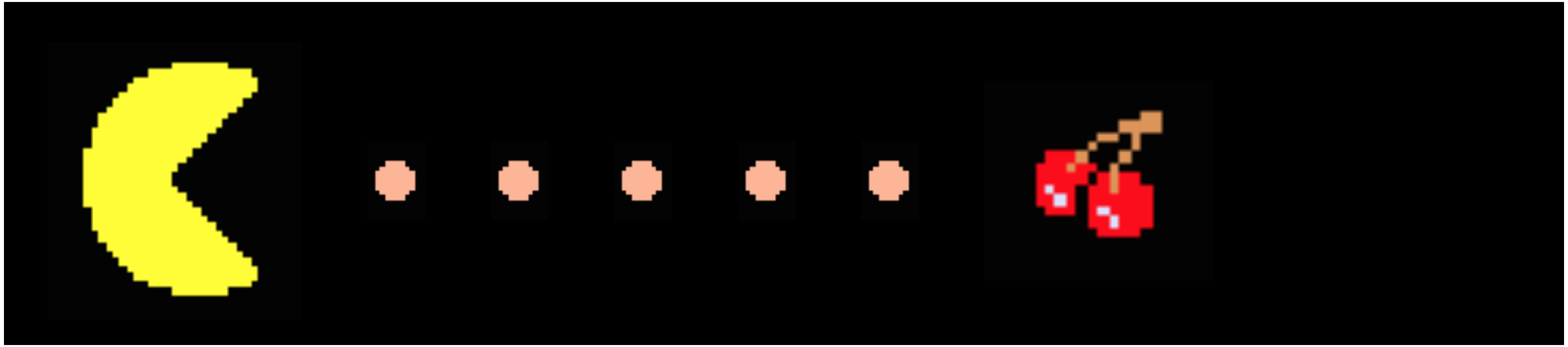
Short-term ("myopic") reward



Long-term reward



vs.



The Price of Greed

Long-term Value

- (Usually) don't want to find best single-step reward
- Want some notion of long-term *aggregate* reward
- **Definition: *Value function:***

$$V := r_1 + r_2 + r_3 + r_4 + \ldots$$

Long-term Value: Mathematical Aside

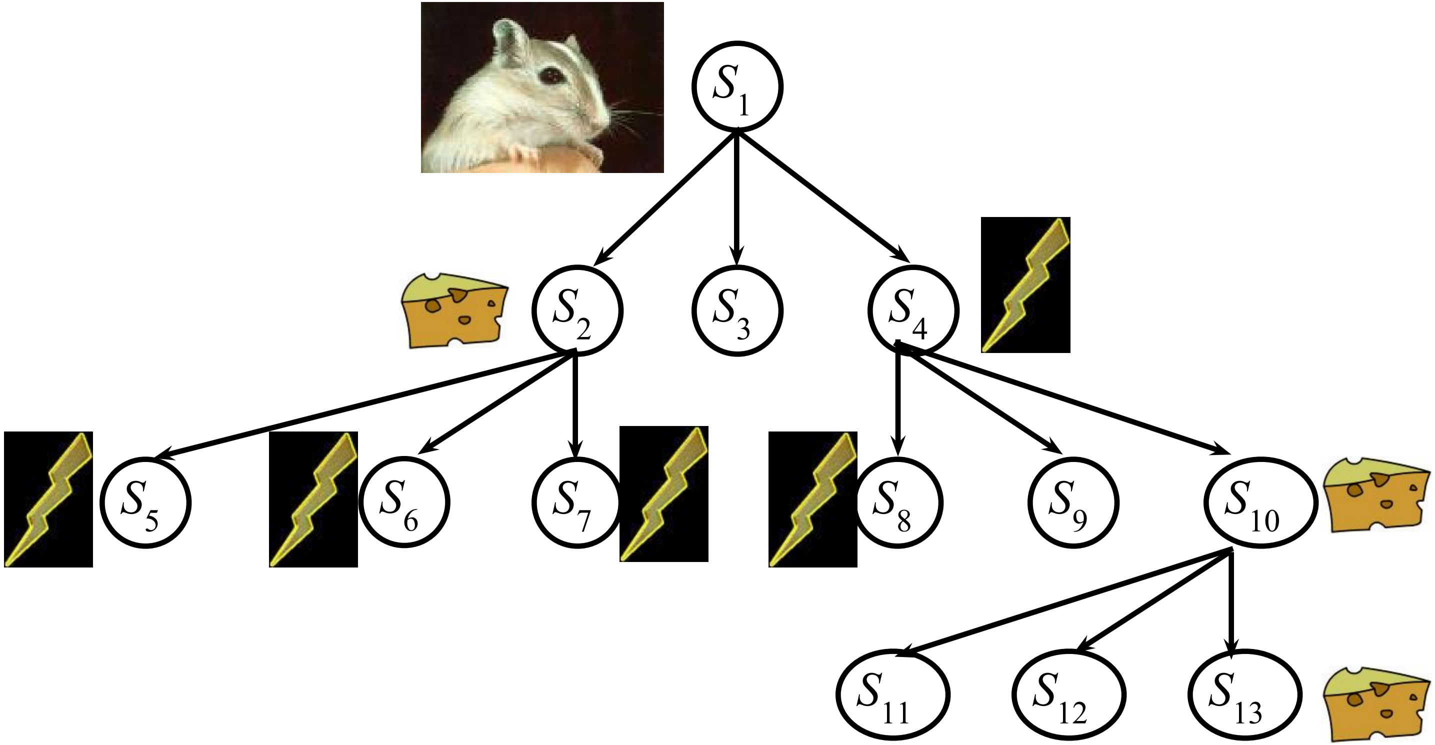
- Infinite sum can diverge (duh)

- Three usual "methods" to fix this:
  - Sum only over a fixed, finite horizon: $V := r_1 + r_2 + r_3$
  - Average: $lim_{T\text{->}inf}\ 1/T\ (V := r_1 + r_2 + r_3 + r_4 + \ldots)$
  - Infinite discounting:

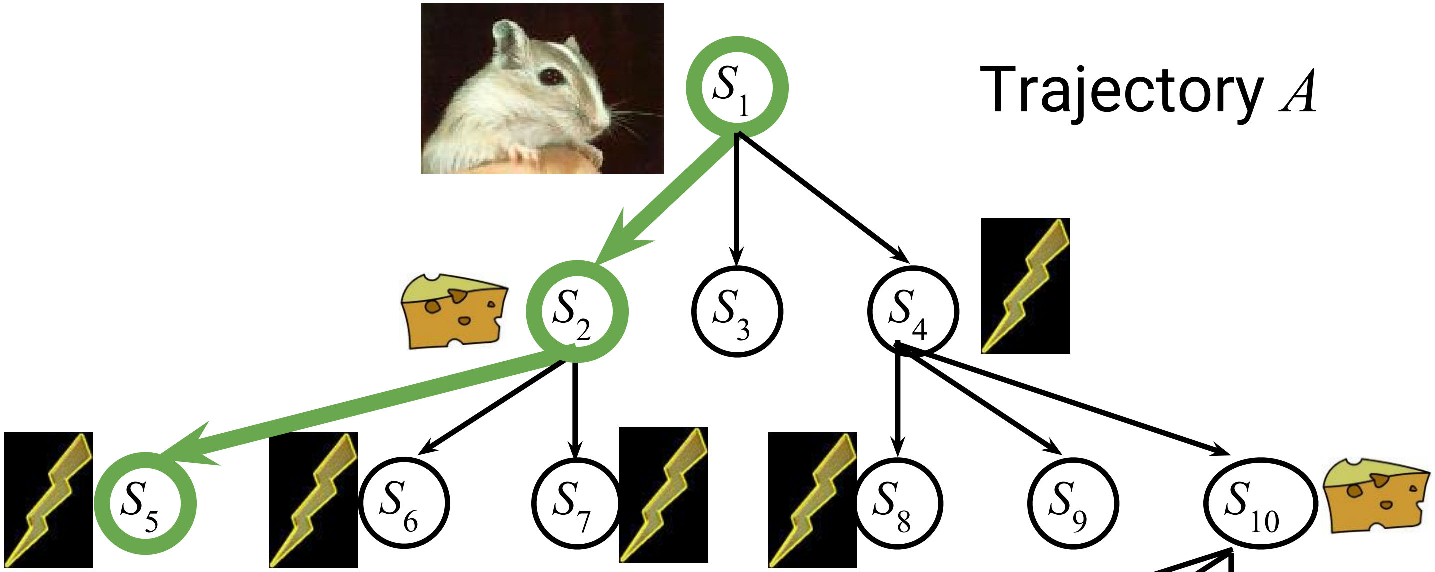
    $$V := \gamma r_1 + \gamma^2 r_2 + \gamma^3 r_3 + \gamma^4 r_4 + \ldots$$

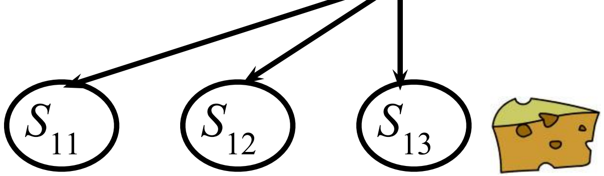
    for $0 <= \gamma < 1$

Life Trajectories

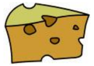


Life Trajectories

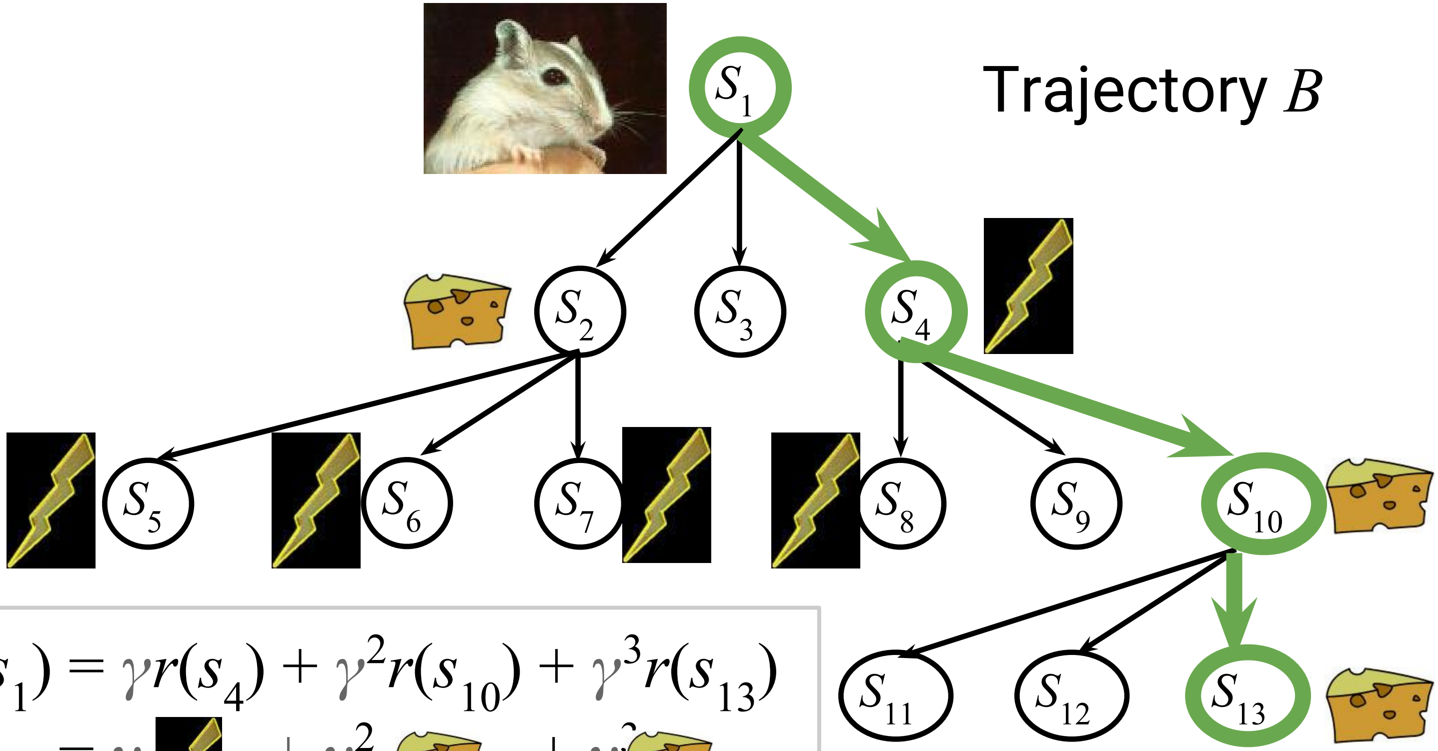




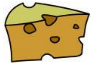
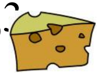
$$V^A(s_1) = \gamma r(s_2) + \gamma^2 r(s_5)$$

$= \gamma$  $+ \gamma^2$ 

Life Trajectories

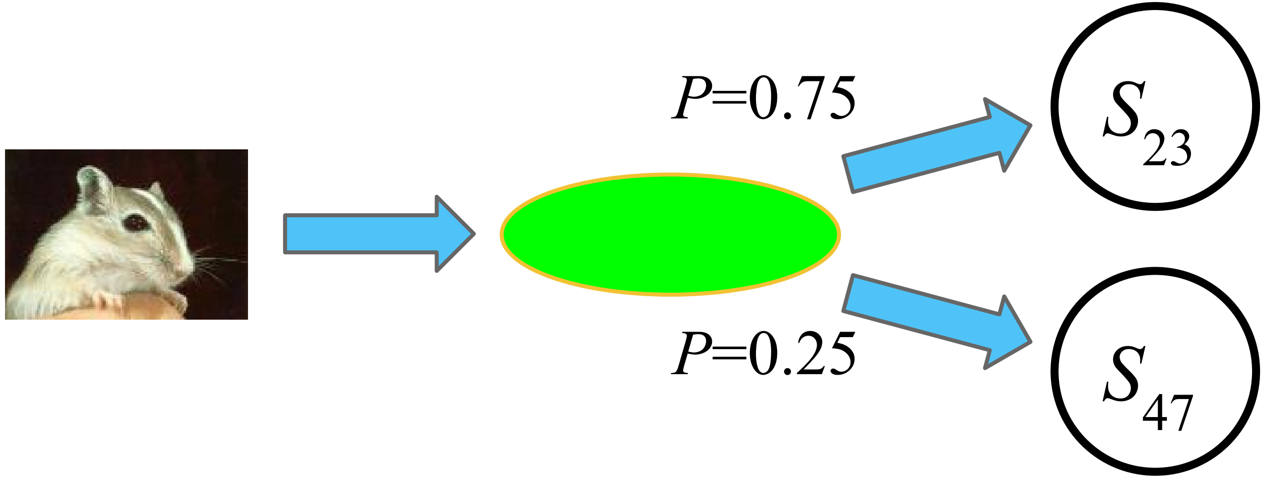


$$V^B(s_1) = \gamma r(s_4) + \gamma^2 r(s_{10}) + \gamma^3 r(s_{13})$$

$= \gamma$  $+ \gamma^2$  $+ \gamma$ 

Final Key Element: Uncertainty

- Life is uncertain
- Actions have unpredictable consequences...

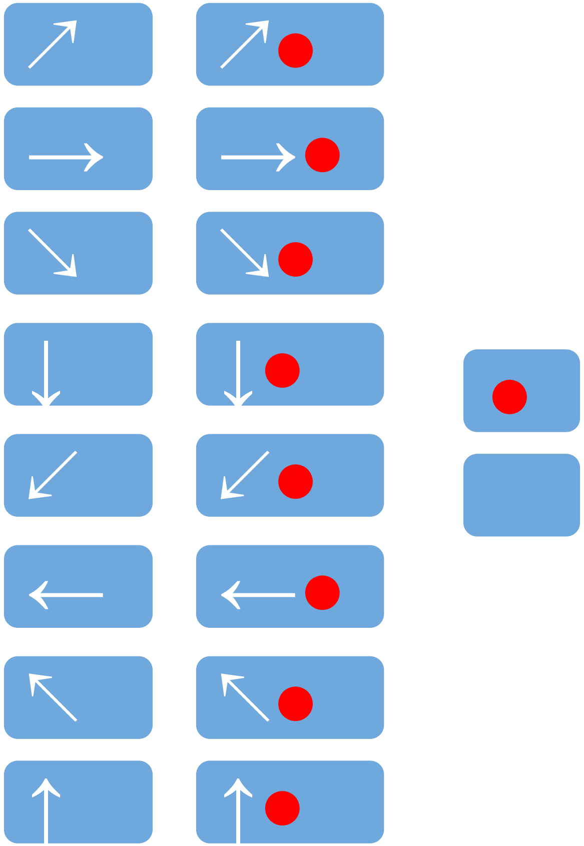


Formalism: The Markov Decision Process (MDP)

- **Definition: MDP** $M = <S, A, T, R>$
- State space: $S = \{s_1, s_2, \ldots, s_N\}$ (possibly infinite)
- Action space: $A = \{a_1, a_2, \ldots., a_k\}$ (possibly infinite)
- Transition function: $T : S \times A \times S \rightarrow [0, 1]$
  - Set of Markov chains, indexed by action: $T_a(s_{t+1} \mid s_t)$
- Reward function: $R : S \rightarrow \mathbb{R}$

$$|S| = N \approx 2^{84,672}$$

Atari Platform, State space [Mnih et al., Nature(518) 2015.]



$$K = |A| = 18$$

Atari Platform, Action space [Mnih et al., Nature(518) 2015.]

Mnih et al.: Final MDP formulation details

- Rewards: change in score
  - Clipped to {-1, 0, +1}
- Transition function: Atari emulator + game

Learning How to Act

- Now we know how to *represent the world* (MDP)
- How does Mac *choose how to act*?
- **Definition:** *Policy*
    - Mapping from states to actions: $\pi : S \to A$
- **Learning Problem:**

Given experience in MDP M, find a (near) optimal policy π^*

Learning is Hard...

- If you had examples of optimal actions, learning π^* would be trivial
- All you see are *histories* (a.k.a., *trajectories*)
  - $(s_{t1}, a_{t1}, r_{t1}), (s_{t2}, a_{t2}, r_{t2}), (s_{t3}, a_{t3}, r_{t3}), \ldots$
- Can calculate *value* of each trajectory, but…
- Which action(s) helped and which hurt it?
- ***Credit assignment problem***

To Model or Not to Model?

- Two fundamental approaches to RL
- “Model based”
  - Learn M = <S, A, T, R> (i.e., learn T and R)
  - Apply a planning algorithm to find optimal π^* for M
    - Poly time in $|S \times A|$
  - See, for example, E^3 **algorithm** (Kearns & Singh)

To Model or Not to Model?

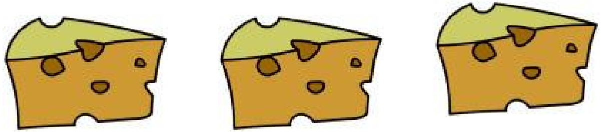
- Two fundamental approaches to RL
- “Model free”
  - Skip learning the MDP model itself
  - Learn π* directly or indirectly
  - Bonus: Don’t necessarily need to touch all state/action pairs

Q

- Possible to learn π directly (see, e.g., *policy gradient* methods)
- Often use a proxy function: Q
- **Definition:** $Q(s, a)$ is value of taking action a at s and then acting according to current policy thereafter
- Think of it as "testing out" action a

Q example

s_1 s_2 s_3



s_4

s_5 s_6 s_7

Current policy

$Q(s_4, \rightarrow)$

then….

Q example

s_1 s_2 s_3 s_4 s_5 s_6 s_7

Current policy

$Q(s_4, \leftarrow)$

then….

Learning Q: Q Learning

- Classic Q learning algorithm:

$$Q_{t+1}(s_t,a_t) = Q_t(s_t,a_t) + \alpha\,(r_{t+1} + \gamma \max_a Q_t(s_{t+1},a) - Q_t(s_t,a_t))$$

- Form of *temporal differencing*
- Modifies estimate of Q by “backing up” experience by one step

Breaking it Down

New guess at value of a_t in s_t

Old guess at value of a_t in s_t

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha\,\big(\underbrace{r_{t+1} + \gamma \max_a Q_t(s_{t+1}, a)}_{\text{What } Q(s_t,a_t) \text{ “should” be}} - Q_t(s_t, a_t)\big)$$

Hedging your bets

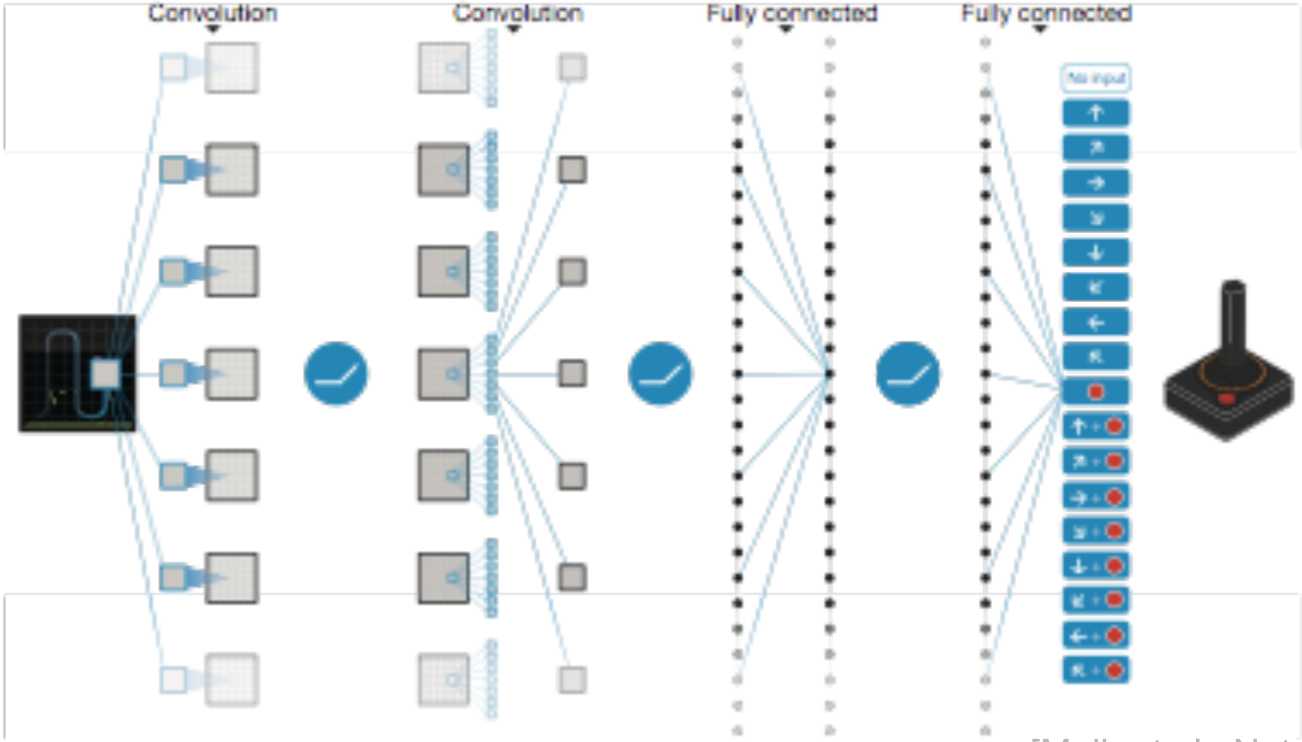
What actually happened when you tried a_t from s_t

What $Q(s_t, a_t)$ “should” be

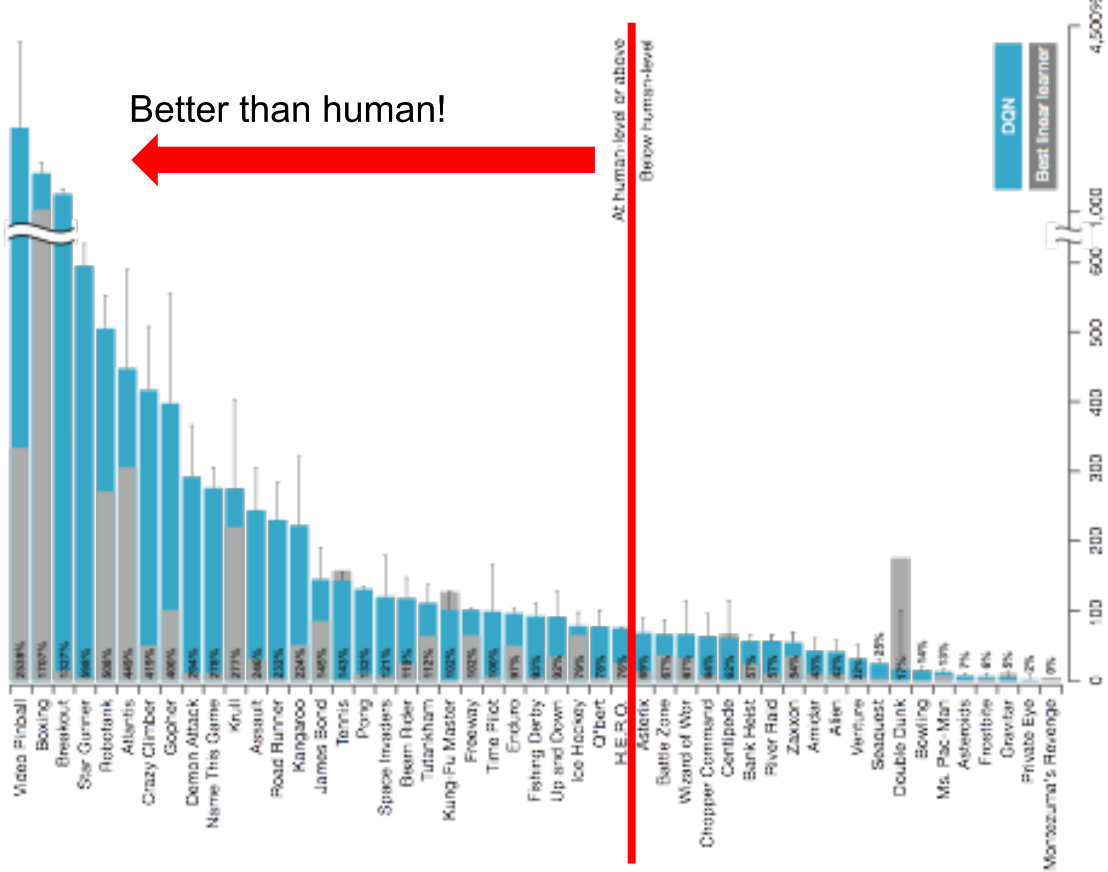
Representing Q

- When $|S \times A|$ is small, can store $Q(s, a)$ as an array
- In Atari world, $|A|$ is small, but $|S|$ is immense
- Enter ***function approximators***
- Replace exact (tabular) Q with approximate $f \approx Q$
- Common choice: Neural network / deep learner
    - C.f., “neuro-dynamic programming”

The Google DeepMind Deep Q Learner



[Mnih et al., Nature(518) 2015.]



[Mnih et al., Nature(518) 2015.]

Hidden State

- Look at some of the failures
  - Wizard of Wor
  - Ms. Pac Man
  - Montezuma's revenge
    - "Adventure" and "Haunted House" don't even show up in list...
- All have some (or a lot) ***hidden state***
  - Parts of the real game state can't be seen by the agent
    - Don't appear in pixels on screen
  - Darkness, ghosts' direction, states of doors, etc.

Why Hidden State is Hard (intuitively)

Why Hidden State is Hard (intuitively)

Why Hidden State is Hard (mathematically)

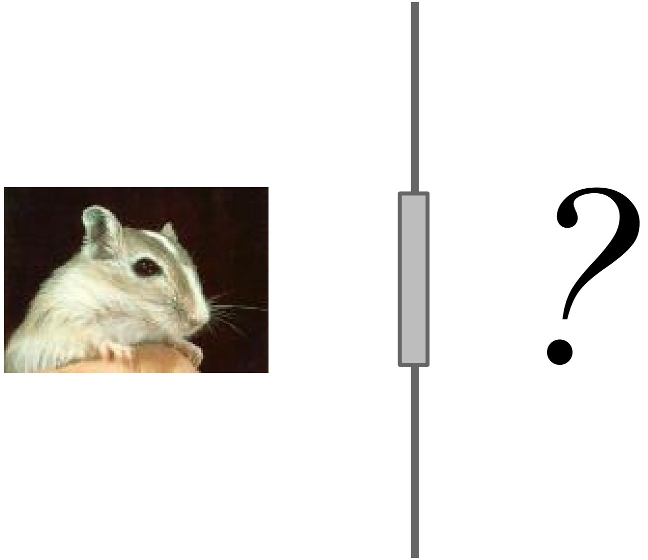
- No longer living in a (nice) MDP
- Now we're in a ***POMDP***
  - Partially observable Markov decision process
- MDP, plus observations and observation function

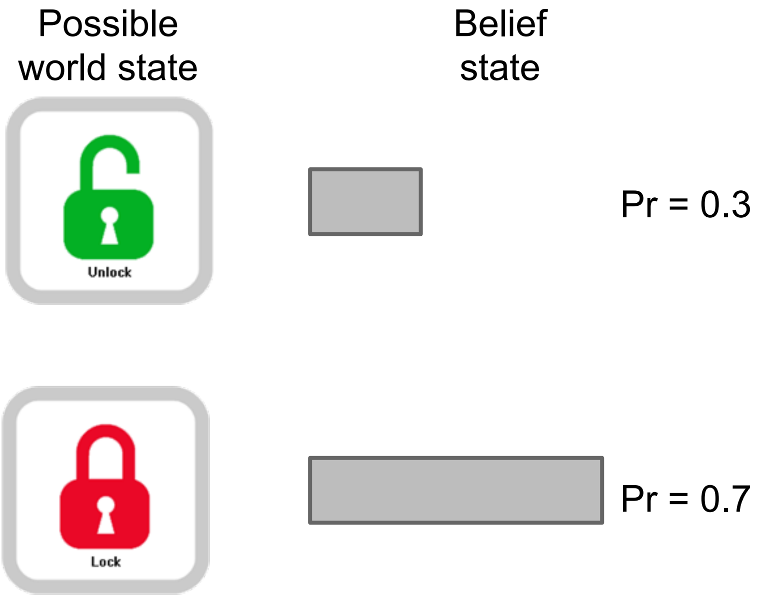
$$P = \langle S, A, T, R, \Omega, O \rangle$$

- *POMDPs are not your friend*

Why POMDPs are Hard

- Have to maintain estimate of probability over every possible hidden state
  - ***Belief state***





Why POMDPs are Hard (cont'd)

- Formally, POMDP is equivalent to *an* MDP…
- … where the state space, S, is a probability simplex of dimension related to the bits of hidden state
    - Called the "belief state MDP"
- Problems that are polynomially solvable for (finite) MDP (e.g., planning) become *uncomputable*
    - Can require unbounded precision in maintaining belief state

Onward and Forward

- Planning and RL in POMDPs is (very big) area of active research
- Lots of progress, but remains super-hard
- Can do real things with, e.g., robot navigation, though
- Sridhar Mahadevan will tell you much more than I could...

Thank you!

Questions?