Markov Decision Processes (cont'd)

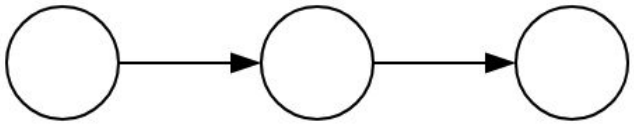
Dr. Angelica Lim
Assistant Professor
School of Computing Science
Simon Fraser University, Canada
Oct. 22, 2024

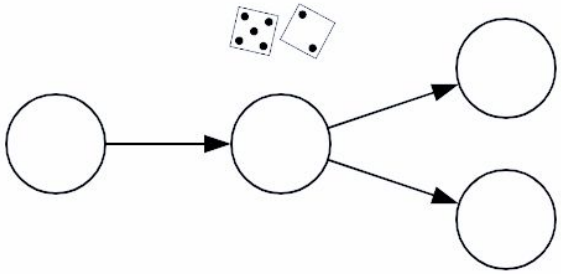
Slide Credits:Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley (ai.berkeley.edu).
Pearson 2021

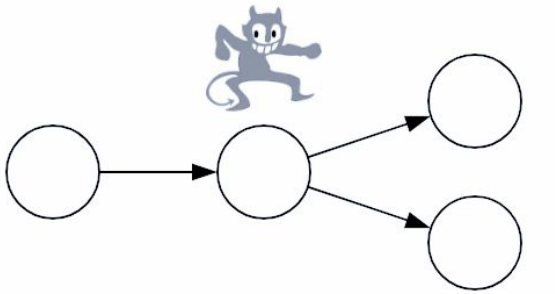
Course Overview

Week 1 : Getting to know you
Week 2 : Introduction to Artificial Intelligence
Week 3: Machine Learning I: Basic Supervised Models (Classification)
Week 4: Machine Learning II: Supervised Regression, Classification and Gradient Descent, K-Means
Week 5: Machine Learning III: Neural Networks and Backpropagation
Week 6 : Search
Week 7 : Markov Decision Processes
Week 8 : Midterm
Week 9 : Reinforcement Learning
Week 10 : Games
Week 11 : Hidden Markov Models and Bayesian Networks
Week 12 : Constraint Satisfaction Problems
Week 13 : Ethics and Explainability

State-based models







Search problems: when you have an environment with no uncertainty, ie. perfect information. But realistic settings are more complex

Markov decision processes (MDPs) handle situations with randomness, e.g. Blackjack

Game playing handles tasks where there is interaction with another agent. Adversarial games assume an opponent, e.g. Chess

Today's Topics

Markov Decision Processes (cont'd)

Ch. 17.2.1 - Value Iteration (Policy extraction)

Ch. 17.3.0 - Bandit Problems

Reinforcement Learning

Ch. 22.1 - Learning from Rewards

Ch. 22.2 - Passive Reinforcement Learning

- 22.2.1 - Direct Utility Estimation (aka Direct Estimation/Evaluation)
- 22.2.3 - Temporal Difference Learning

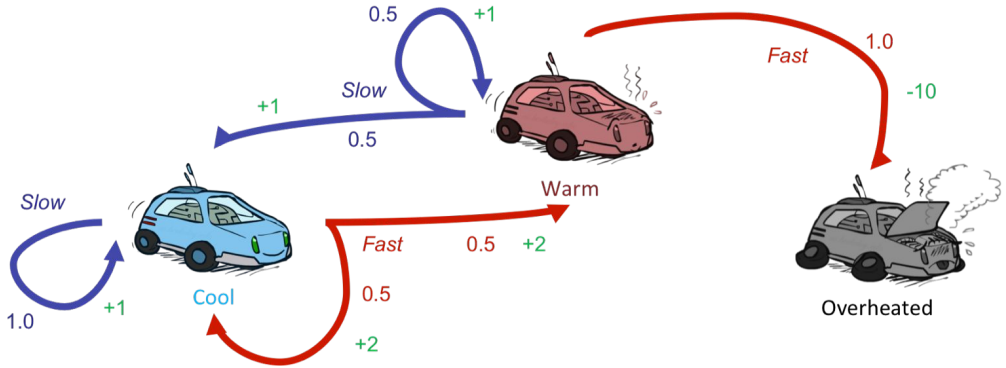
Ch. 22.3 - Active Reinforcement Learning

- 22.3.3 - Temporal Difference Q-Learning

Recap: Value Iteration

V_2	3.5	2.5	0
V_1	2	1	0
V_0	0	0	0

Keep iterating until these values converge



Assume no discount!

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

[Weekly Activity] Markov Decision Processes

Angelica Lim

Oct 22 at 5:17pm

135 144

Upload a photo of your work on the following question.

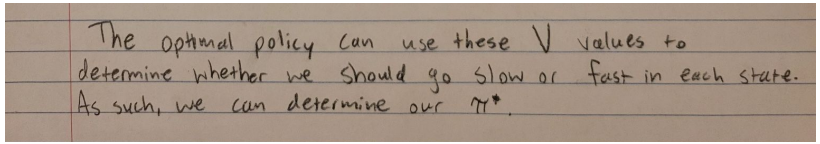
At the end of last class, we looked at the following example, which assumes no discount. Recall that the V(s) is the maximum Q(s,a) over all **actions**.

For example,

- Q(cool, **fast**) = T(cool, fast, cool)(2+0) + T(cool, fast, warm)(2+0) = 0.5*2+0.5*2 = 2
- Q(cool, **slow**) = T(cool, slow, cool)(1) = 1
- max(Q(cool,**fast**), Q(cool,**slow**)) => 2

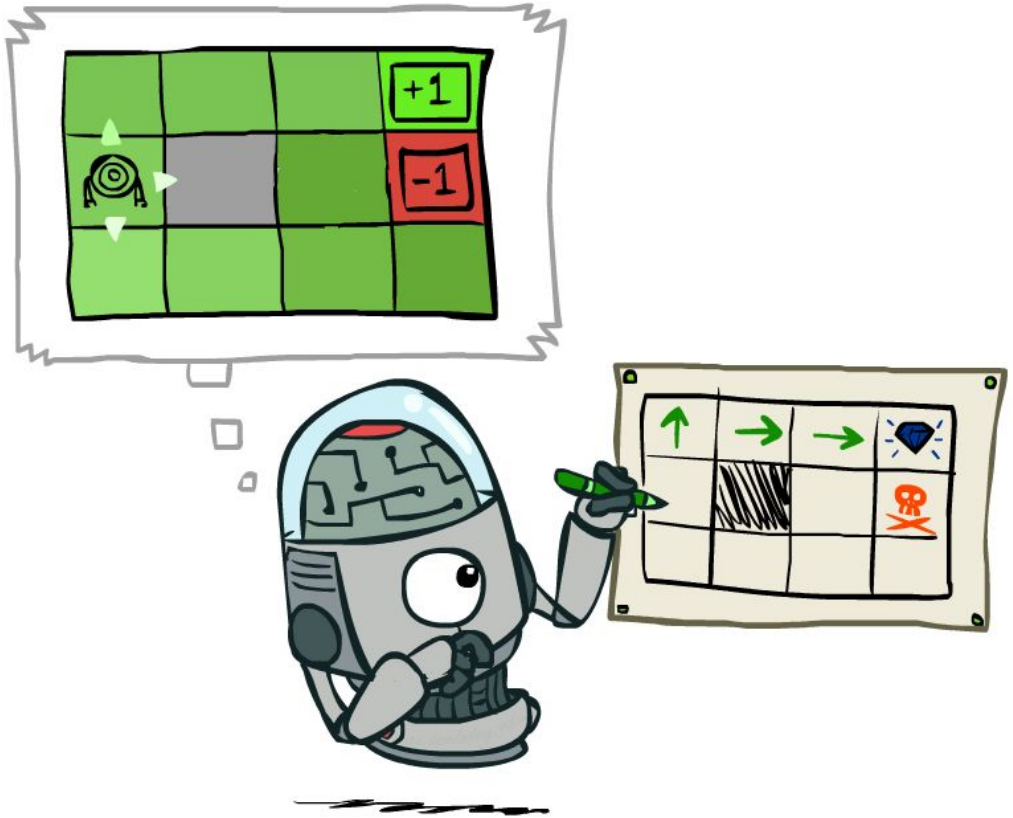
V1(cool) = 2

Show your work in calculating the remaining 5 values. Once we find these values, how could they be useful for coming up with an optimal policy?



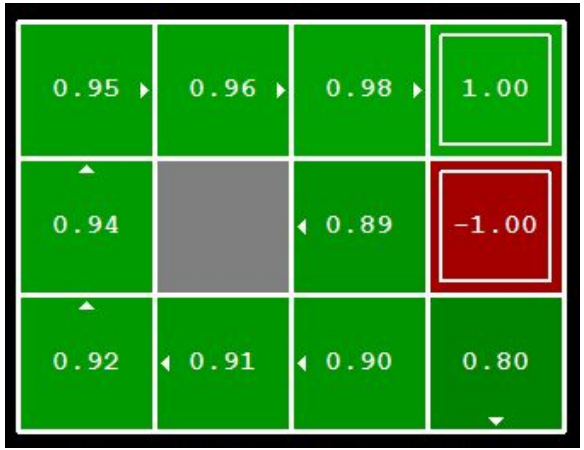
Credit: A. Arani

Policy Extraction



Policy Extraction: Computing Actions from Values

Let's imagine we have the optimal values V*(s)



- How should we act?
  - It's not obvious!
- We need to do one more mini iteration

This is called policy extraction, since it gets the policy implied by the values

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

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

Policy Extraction: Computing Actions from Values

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

s	a	s'	T(s,a,s')	R(s,a,s')
	Slow		1.0	+1
	Fast		0.5	+2
	Fast		0.5	+2
	Slow		0.5	+1
	Slow		0.5	+1
	Fast		1.0	–10
	(end)		1.0	0

This tells us that the action of going fast will maximize the value obtained on all future actions.

V_2

3.5	2.5	0

π*(cool) = argmax_a{slow,fast}

a = slow:
T(cool,slow,cool)[R(cool,slow,cool)+3.5]=1*[1+3.5]
+T(cool,slow,warm)[R(cool,slow,fast)+2.5]=0 4.5

a = fast:
T(cool,fast,cool)[R(cool,fast,cool)+3.5] = 0.5[2+3.5]
+T(cool,fast,warm)[R(cool,fast,warm)+2.5] = 0.5[2+2.5] 5

Policy  Fast

 …

Policy Extraction: Computing Actions from Q-Values

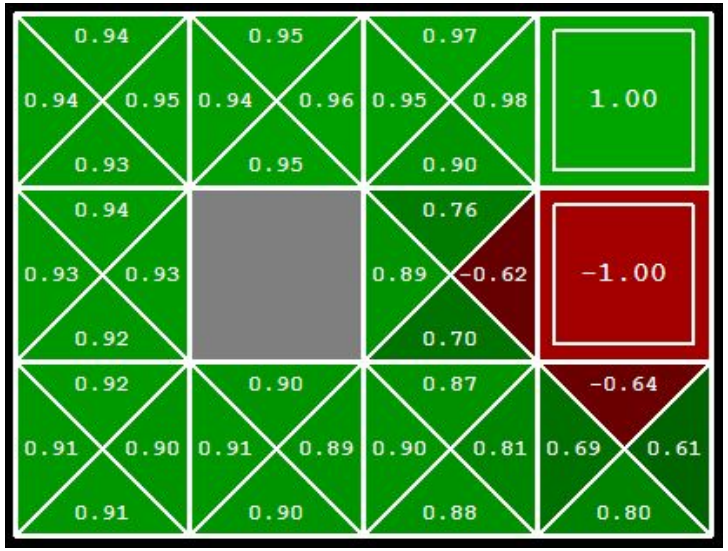
Let's imagine we have the optimal q-values.

How should we act?

- Completely trivial to decide!

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

Important lesson: actions are easier to select from q-values than values!



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:

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

Why are Q-values more useful?

- 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:

Deep Q-learning is better for neural networks

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

Alternative to Value Iteration

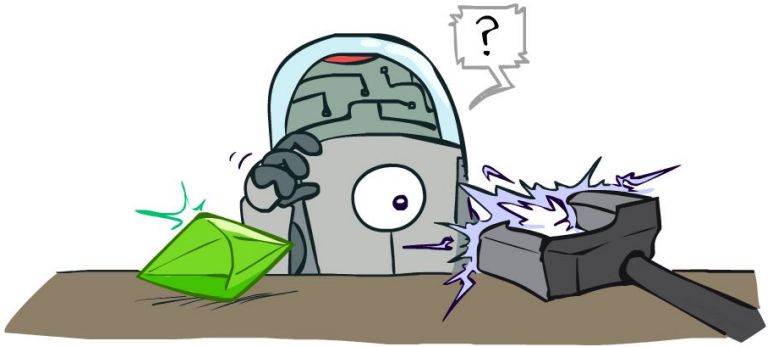
There is a faster algorithm that can also solve an MDP, which finds a policy faster than value iteration. The basic idea:

- Step 1: Policy evaluation: calculate values for some fixed policy (not optimal values!) until convergence
- Step 2: Policy improvement: update policy using with resulting converged (but not optimal!) values as future values
- Repeat steps until policy converges

This is policy iteration

- It's still optimal!
- Can converge (much) faster under some conditions

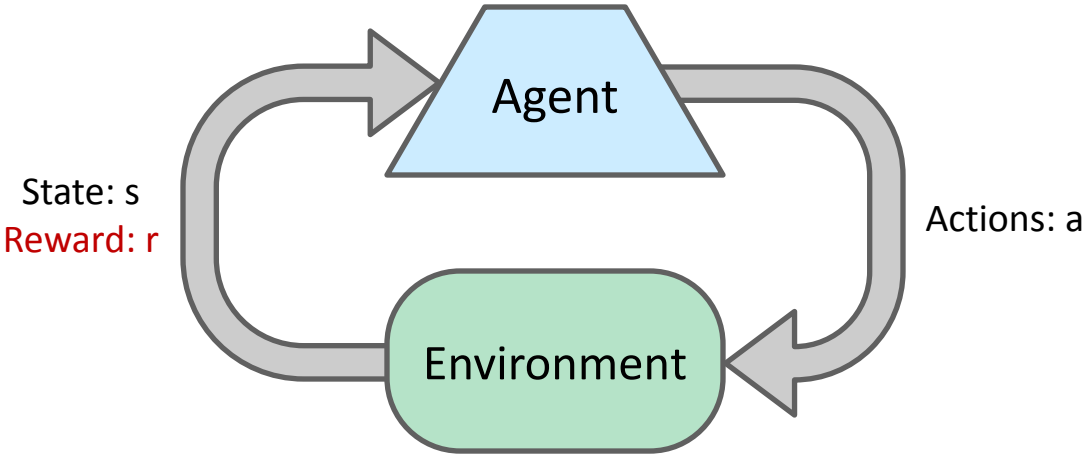
Reinforcement Learning



Dr. Angelica Lim
Assistant Professor
School of Computing Science
Simon Fraser University, Canada
Oct. 22, 2024

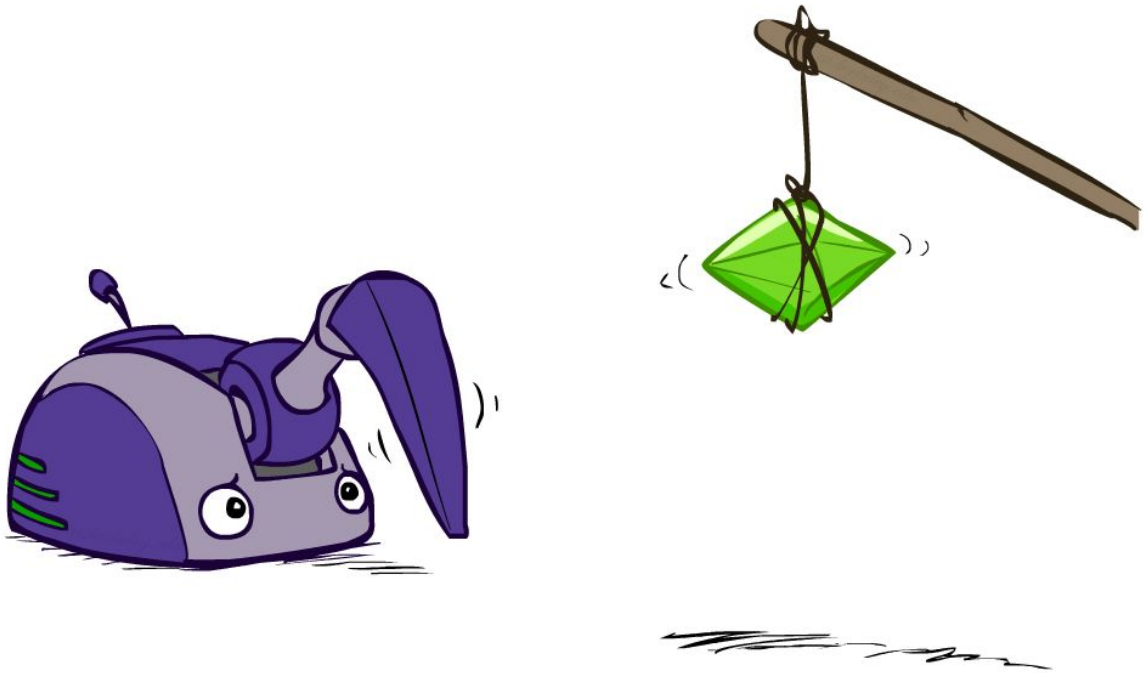
Slide Credits:Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley (ai.berkeley.edu).

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 Applications



Example: Learning to Walk

Initial

[Kohl and Stone, ICRA 2004]

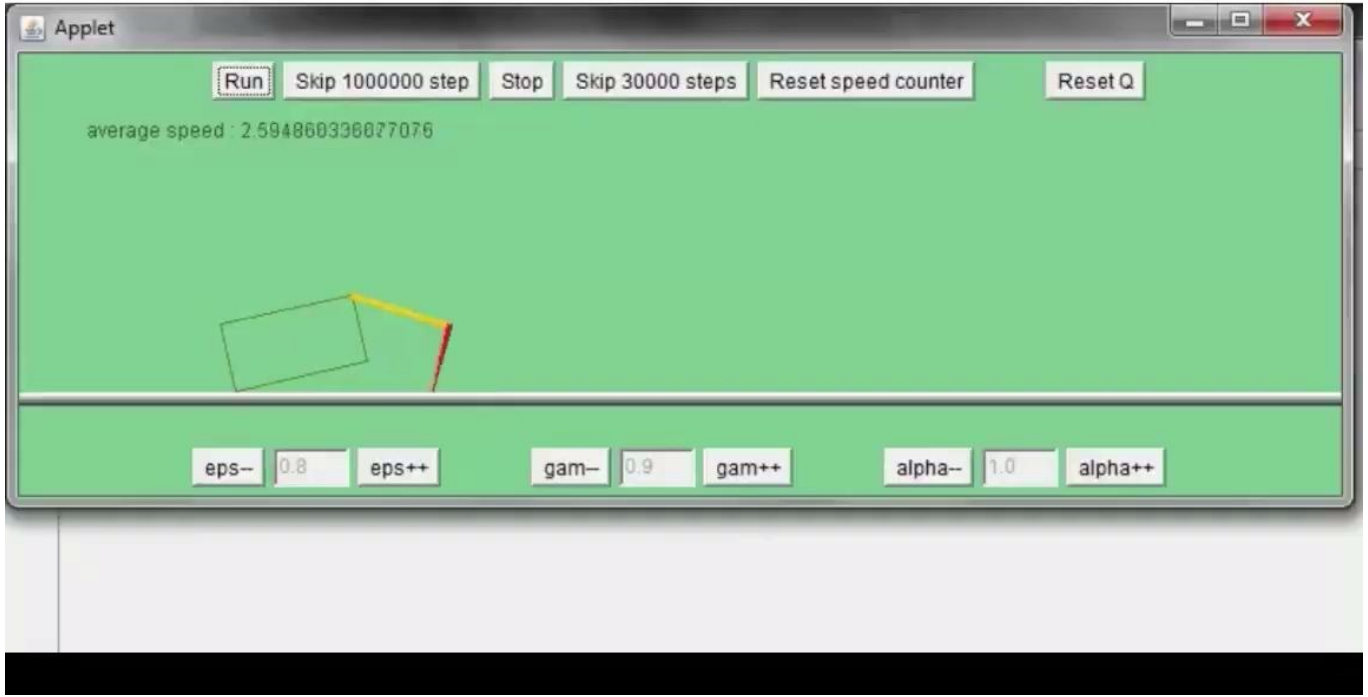
Example: Learning to Walk

Finished

[Kohl and Stone, ICRA 2004]

Example: Breakout (DeepMind)

The Crawler!


Applet
Run
Skip 1000000 step
Stop
Skip 30000 steps
Reset speed counter
Reset Q
average speed : 2.594860336077076
eps--
0.8
eps++
gam--
0.9
gam++
alpha--
1.0
alpha++

Reinforcement Learning

- Still assume a Markov decision process (MDP):
  - A set of states $s \in S$
  - A set of actions (per state) $A(s)$
  - A transition 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 explore new states and actions to discover how the world works

Recall: Value Iteration

s	a	s'	T(s,a,s')	R(s,a,s')
	Slow		1.0	+1
	Fast		0.5	+2
	Fast		0.5	+2
	Slow		0.5	+1
	Slow		0.5	+1
	Fast		1.0	–10
	(end)		1.0	0

P(s'|s,a)

Value iteration required *knowing* the transition function probabilities.

What if you don't know these probabilities?

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

Assume no discount!

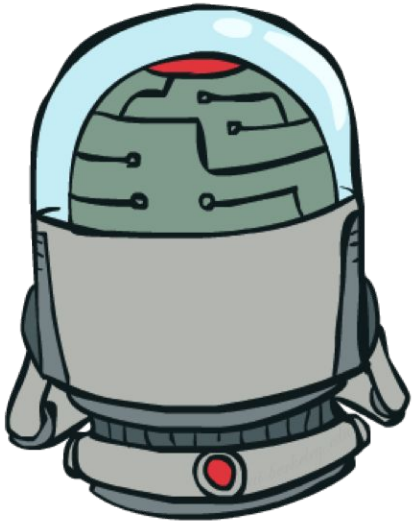
How would you test that a slot machine is not rigged?

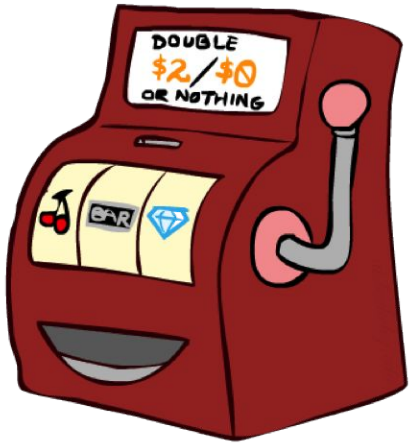
https://www2.gov.bc.ca/assets/gov/sports-recreation-arts-and-culture/gambling/gambling-in-bc/stds-tech-gaming-tgs1.pdf

One-Arm Bandits

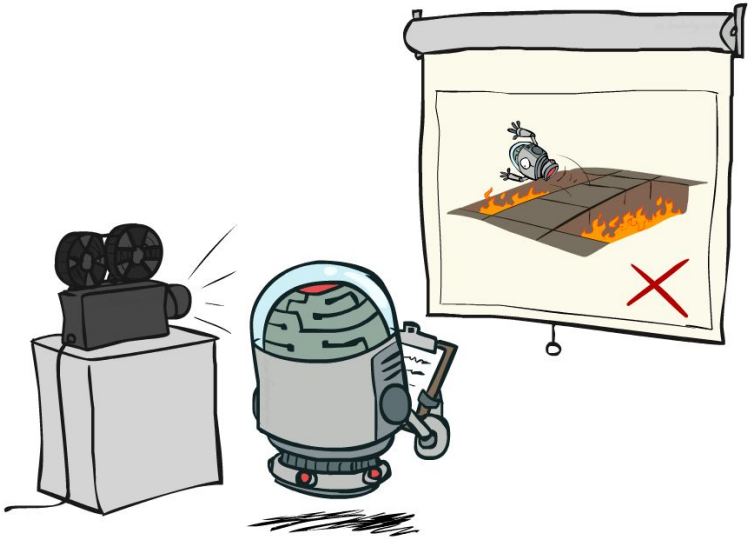
If you have an optimal policy for working with one-arm bandits, does it mean that the machine is rigged?

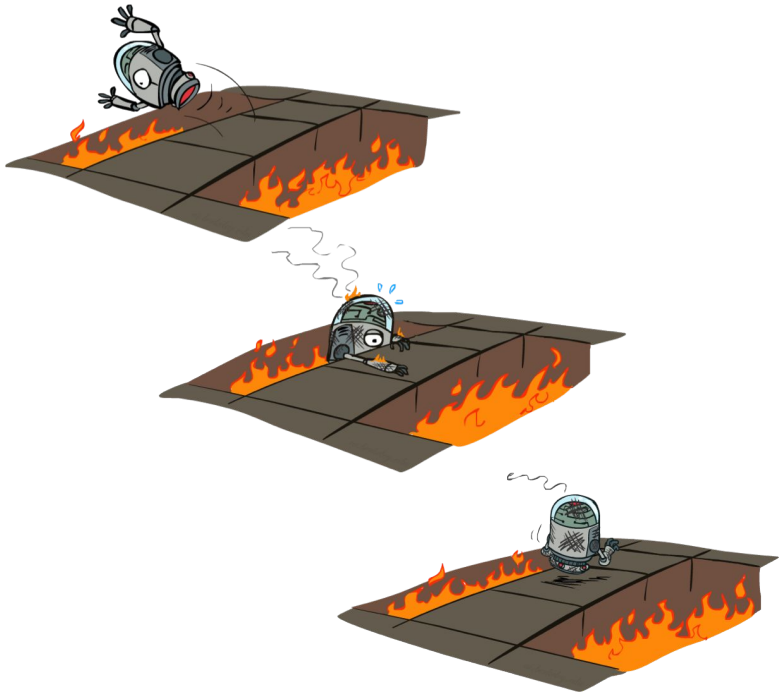
More specifically, if you have an optimal policy that yields a substantial amount of cash, does it mean the machine is rigged?





Passive vs Active RL

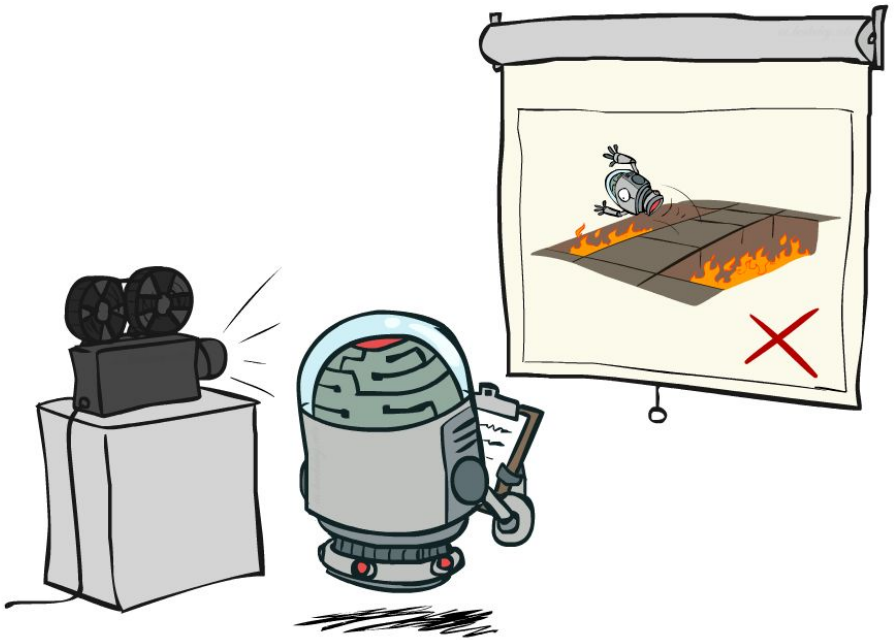




Passive (fixed π)

Active (changing π)

Passive Reinforcement Learning



Passive (fixed π)

Approaches to reinforcement learning

There are hundreds of approaches. We will learn basic approaches for each of the following families:

1. **Model-based**: Learn the model, solve it, execute the solution
2. **Model-free**: Learn values from experiences, use to make decisions
   - Direct evaluation
   - Temporal difference learning
   - Q-learning

Passive RL

Passive RL

Passive RL

Active RL

Passive Reinforcement Learning

Model-Based Learning

Model-Based Learning

Episode is analagous to epoch?
Somewhat, but there are differences.
Ask Angelica about these differences.

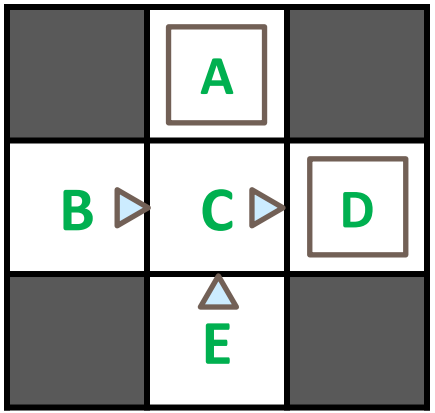
- **Model-Based** Idea:
  - Learn an approximate model based on experiences
  - Solve for values as if the learned model were correct

- **Step 1:** Learn empirical MDP model
  - Count outcomes s' for each s, a
  - Directly estimate each entry in $T(s,a,s')$ from counts
  - Discover each $R(s,a,s')$ when we experience the transition

- **Step 2:** Solve the learned MDP
  - Use, e.g., 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

$T(s,a,s')$

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

$R(s,a,s')$

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

There is randomness in the model. as a result, going to D may result in A being reached.

Example: Expected Age

Goal: Compute expected age of CMPT 310 students

Known P(A)

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

Is the sampling being done with or without replacement?

N is the number of samples

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

"Model Based": estimate P(A):

What does N_a represent?
It is the number of occurrences of students with age a.

$$\hat{P}(A=a) = N_a/N$$

$$E[A] \approx \sum_a \hat{P}(a) \cdot a$$

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

"Model Free": estimate expectation

$$E[A] \approx 1/N \sum_i a_i$$

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

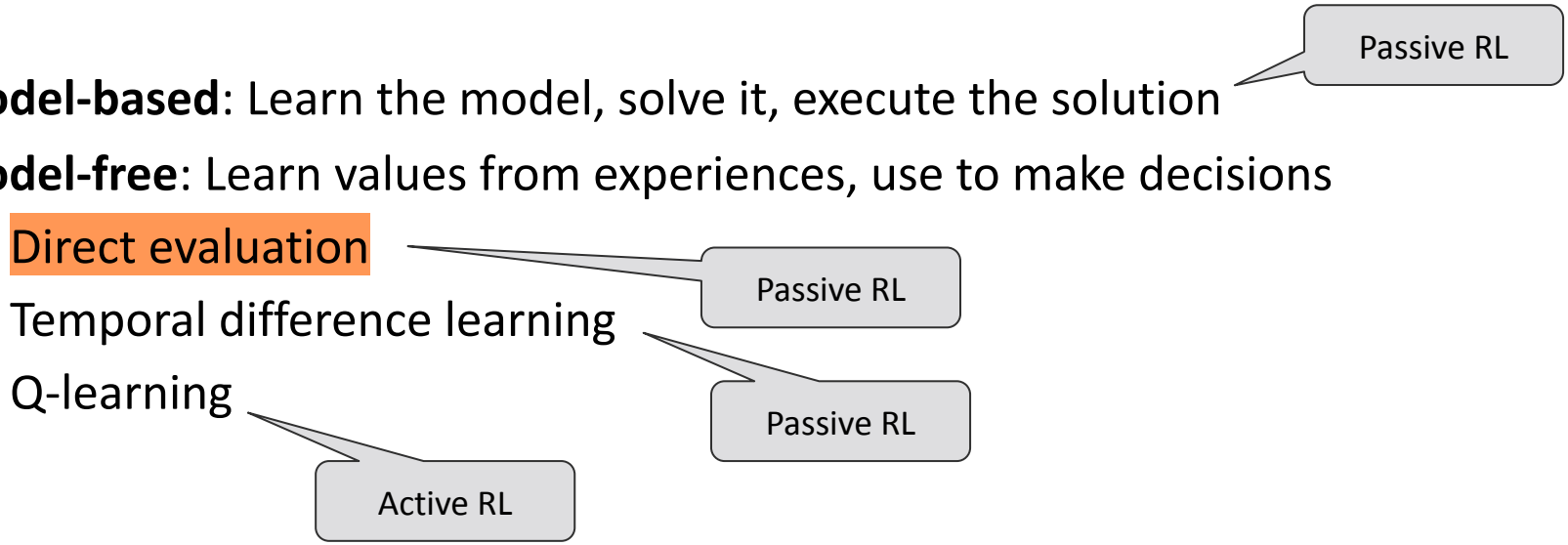
Passive Reinforcement Learning (Model-free)

Direct Evaluation

Approaches to reinforcement learning

There are hundreds of approaches. We will learn basic approaches for each of the following families:

1. **Model-based**: Learn the model, solve it, execute the solution
2. **Model-free**: Learn values from experiences, use to make decisions



Direct evaluation

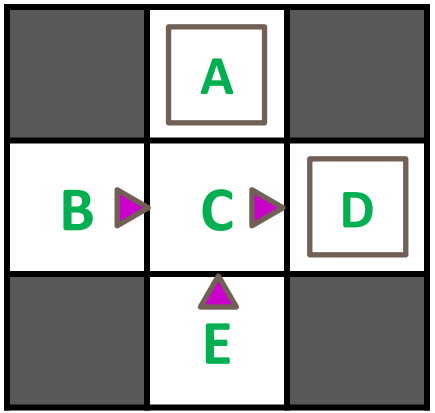
- Goal: Estimate $V^{\pi}(s)$, i.e., expected total discounted reward from s onwards

- Idea:
  - Use ***returns***, the *actual* sums of discounted rewards from s
  - Average over multiple trials and visits to s

- This is called ***direct evaluation*** (or direct utility estimation)

Example: Direct Estimation of Values

Input Policy π



Assume: γ = 1

Observed Episodes (Training)

Episode 1

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 2

B, east, C, -1
C, east, D, -1
D, exit, x, +10

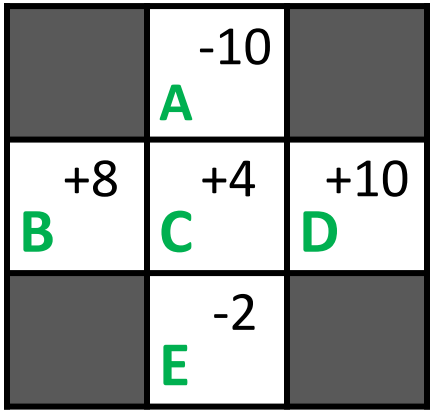
Episode 3

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

Episode 4

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

Output Values

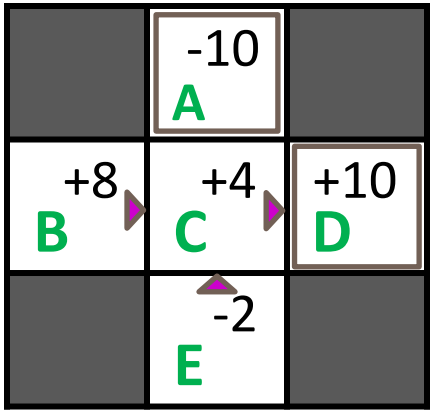


Problems with Direct Estimation

- **What's good about direct estimation?**
  - It's easy to understand
  - It doesn't require any a priori knowledge of *T* and *R*
  - It converges to the right answer in the limit

- **What's bad about it?**
  - Each state must be learned separately (fixable)
  - It ***ignores information about state connections***
  - 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?

E.g., B=at home, study hard
E=at library, study hard
C=know material, go to exam

Passive Reinforcement Learning (Model-free)

Temporal Difference Learning

Approaches to reinforcement learning

There are hundreds of approaches. We will learn basic approaches for each of the following families:

1. **Model-based**: Learn the model, solve it, execute the solution
2. **Model-free**: Learn values from experiences, use to make decisions

Passive RL

Direct evaluation

Temporal difference learning

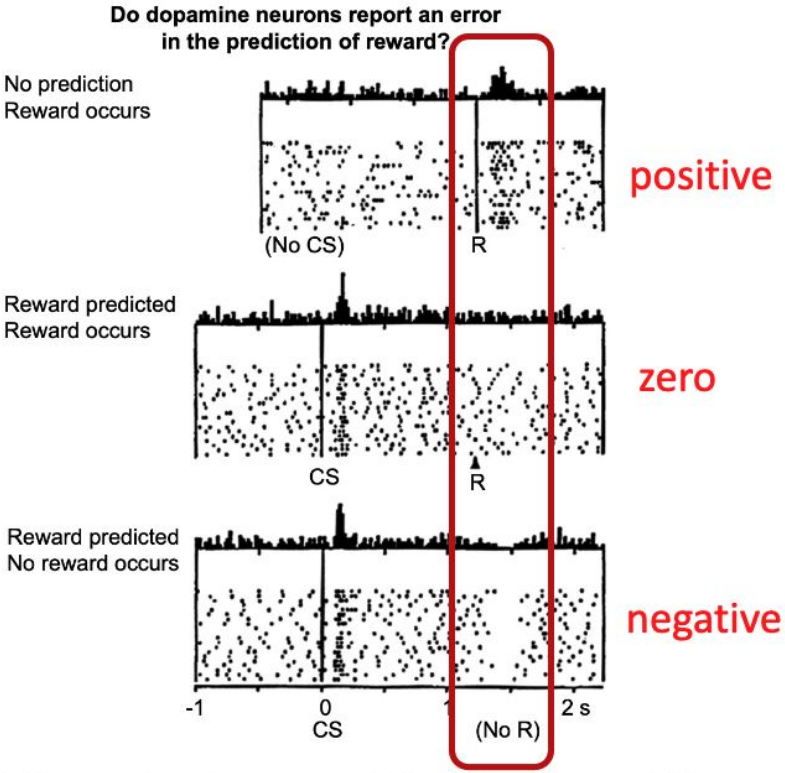
Q-learning

Passive RL

Passive RL

Active RL

TD Learning in the Brain



[A Neural Substrate of Prediction and Reward. Schultz, Dayan, Montague. 1997]

SFU

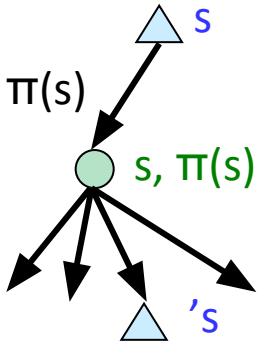
Temporal Difference Learning

Remember SGD and learning from each sample instead of the entire train error?

- **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

Why are we recording the difference in Temporal Difference learning?

- **Temporal difference** learning of values
  - Policy still fixed, still doing evaluation!
  - Move values toward value of whatever successor occurs: **running average**



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

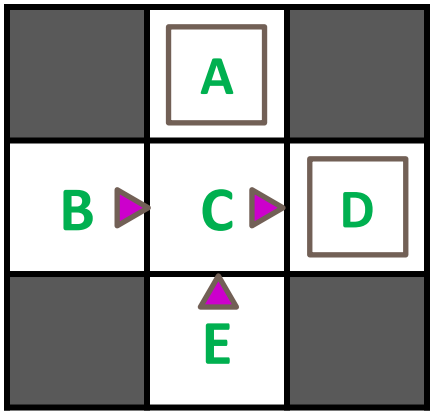
Sample of V(s): $sample = R(s, \pi(s), s') + \gamma V^{\pi}(s')$

Algorithm: TD Value Estimation

- Experience transition *i:* (s_i, a_i, s_i', r_i).
- Compute sampled value "target": $r_i + \gamma V^{\pi}(s_i')$.
- Compute "TD error": $\delta_i = \left(r_i + \gamma V^{\pi}(s_i')\right) - V^{\pi}(s_i)$.
- Update: $V^{\pi}(s_i)$ += $\alpha_i \cdot \delta_i$.

Example: TD Value Estimation

Input Policy π



Assume: γ = 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

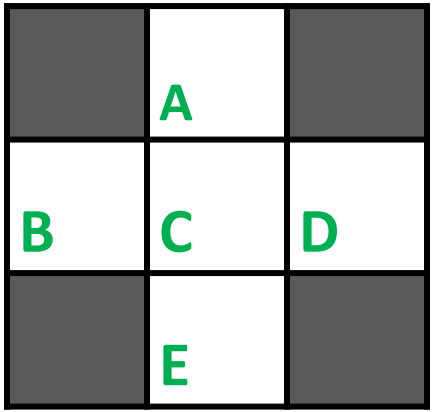
Episode 3

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: TD Value Estimation

- Experience transition *i*: (s_i, a_i, s_i', r_i).
- Compute sampled value "target": $r_i + \gamma V^{\pi}(s_i')$.
- Compute "TD error": $\delta_i = \left(r_i + \gamma V^{\pi}(s_i')\right) - V^{\pi}(s_i)$.
- Update: $V^{\pi}(s_i)$ += $\alpha_i \cdot \delta_i$.

Ask Angelica about this table.
Why is the value for row 4 -2?
Should it not be -1?

What is r_i?
r_i is the reward from transitioning to state s' from s at pass i

Assume alpha, the learning rate, is 1

Assume discount factor is 1

s	V(s)
A	0
B	-1
C	-1
D	10
E	0

i	s	a	s'	r	target value	v^{pi}(s_i)	δ
1	B	E	C	-1	-1 + 0		-1
2	C	E	D	-1	-1 + 0		-1
3	D	Exit	_	10	10 + 0		10
4	B	E	C	-1	-1 - 1 = -2		-1
5	C	E	D	-1			
6							
7							

B, east, C, -1
C, east, D, -1
D, exit, x, +10

B, east, C, -1
C, east, D, -1
D, exit, x, +10

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

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

Example: TD Value Estimation

- Experience transition *i*: (s_i, a_i, s_i', r_i).
- Compute sampled value "target": $r_i + \gamma V^{\pi}(s_i')$.
- Compute "TD error": $\delta_i = \left(r_i + \gamma V^{\pi}(s_i')\right) - V^{\pi}(s_i)$.
- Update: $V^{\pi}(s_i)$ += $\alpha_i \cdot \delta_i$.

For the fifth row, should the value for delta not be 10?

s	V(s)
A	0
B	-1
C	9
D	10
E	8

i	*s*	*a*	*s'*	*r*			*δ*
1	B	east	C	-1	-1 + 0	0	-1
2	C	east	D	-1	-1 + 0	0	-1
3	D	exit	---	10	10 + 0	0	+10
4	B	east	C	-1	-1 + -1	-1	-1
5	C		D	-1	-1 + 10	-1	+9
6	D	exit	---	10	10 + 0	10	0
7	E	north	C	-1	-1 + 9	0	+8

B, east, C, -1
C, east, D, -1
D, exit, x, +10

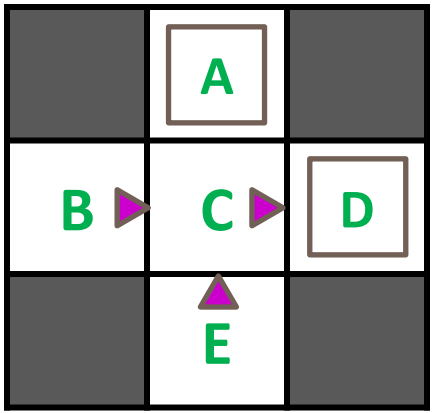
B, east, C, -1
C, east, D, -1
D, exit, x, +10

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

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

Example: TD Value Estimation

Input Policy π



Assume: γ = 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

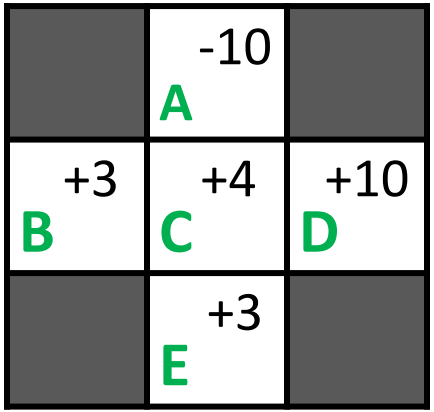
Episode 3

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

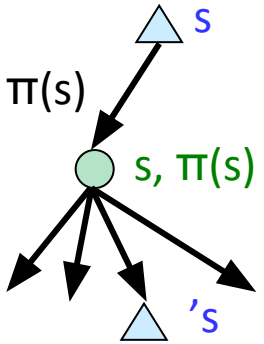
Output Values



Temporal Difference Learning

- Passive setting (fixed policy π)

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



- Modifications:
  1. Don't know T or R; estimate expectation from samples!

$$V^{\pi}(s) = \frac{1}{N} \sum_{i} \left[r_i + \gamma V^{\pi}(s_i') \right]$$

  2. Update V(s) after each transition (s,a,s',r) using running average.

  3. Decay older samples as new ones come in.

Running averages

How do you compute the average of 1, 4, 7?

- **Method 1:** add them up and divide by N
  - 1+4+7 = 12
  - average = 12/N = 12/3 = 4

- **Method 2:** keep a running sum, or running mean μ_n, and a running count, n:

  $\mu_n = (sum_{n-1} + x_n) / count_n = ((n-1) \cdot \mu_{n-1} + x_n)/n$

  - n=0 $\mu_0=0$
  - n=1 $\mu_1 = (0 \cdot \mu_0 + x_1)/1 = (0 \cdot 0 + 1)/1 = 1$
  - n=2 $\mu_2 = (1 \cdot \mu_1 + x_2)/2 = (1 \cdot 1 + 4)/2 = 2.5$
  - n=3 $\mu_3 = (2 \cdot \mu_2 + x_3)/3 = (2 \cdot 2.5 + 7)/3 = 4$

- Alternate formula:

  $\mu_n = [(n-1)/n]\, \mu_{n-1} + [1/n]\, x_n$ (weighted average of old mean, new sample)

Running averages contd.

What if we use a weighted average with a fixed weight?

- $\mu_n = (1-\alpha)\,\mu_{n-1} + \alpha\, x_n$
- $n=1 \quad \mu_1 = x_1$
- $n=2 \quad \mu_2 = (1-\alpha) \cdot \mu_1 + \alpha x_2 = (1-\alpha) \cdot x_1 + \alpha x_2$
- $n=3 \quad \mu_3 = (1-\alpha) \cdot \mu_2 + \alpha x_3 = (1-\alpha)^2 \cdot x_1 + \alpha(1-\alpha)x_2 + \alpha x_3$
- $n=4 \quad \mu_4 = (1-\alpha) \cdot \mu_3 + \alpha x_4 = (1-\alpha)^3 \cdot x_1 + \alpha(1-\alpha)^2 x_2 + \alpha(1-\alpha)x_3 + \alpha x_4$

- I.e., ***exponential forgetting*** of old values
- μ_n is a convex combination of sample values (weights sum to 1)
- $E[\mu_n]$ is a convex combination of $E[X_i]$ values, hence unbiased

TD as approximate Bellman update

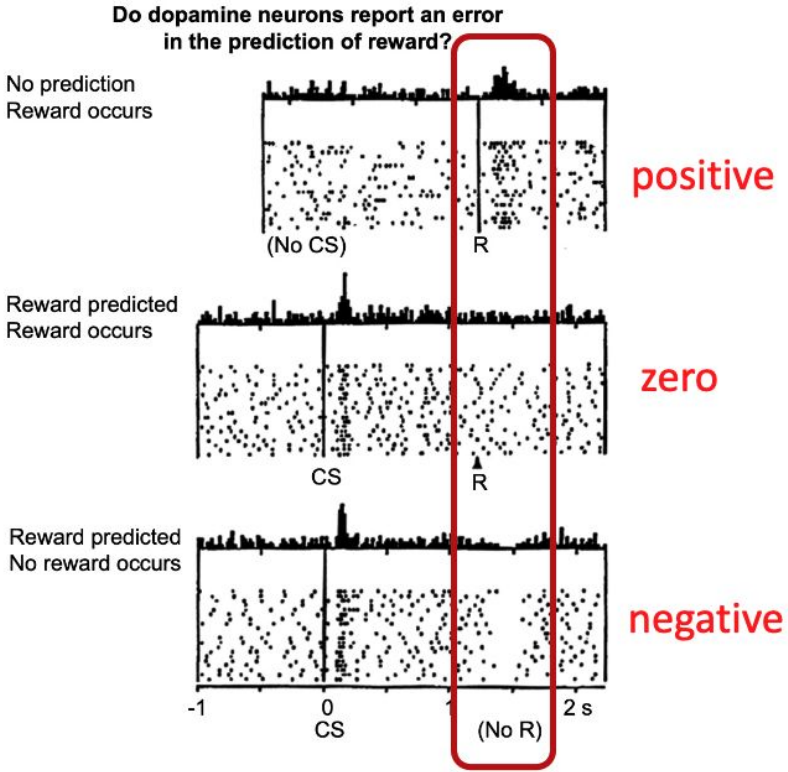
- Experience transition *i*: (s_i, a_i, s_i', r_i).
- Compute sampled value "target": $r_i + \gamma V^{\pi}(s_i')$.
- Compute "TD error": $\delta_i = \left(r_i + \gamma V^{\pi}(s_i')\right) - V^{\pi}(s_i)$.
- Update with TD learning rule:
  - $V^{\pi}(s_i) \leftarrow V^{\pi}(s_i) + \alpha \cdot \delta_i$.
  - $V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha \cdot [\text{target} - V^{\pi}(s)]$
  - $V^{\pi}(s) \leftarrow (1-\alpha) \cdot V^{\pi}(s) + \alpha \cdot \text{target}$
  - α is the ***learning rate***

TD Learning Happens in the Brain!

- Neurons transmit *Dopamine* to encode reward or value prediction error:

$$\delta_i = \left(r_i + \gamma V^{\pi}(s_i')\right) - V^{\pi}(s_i).$$

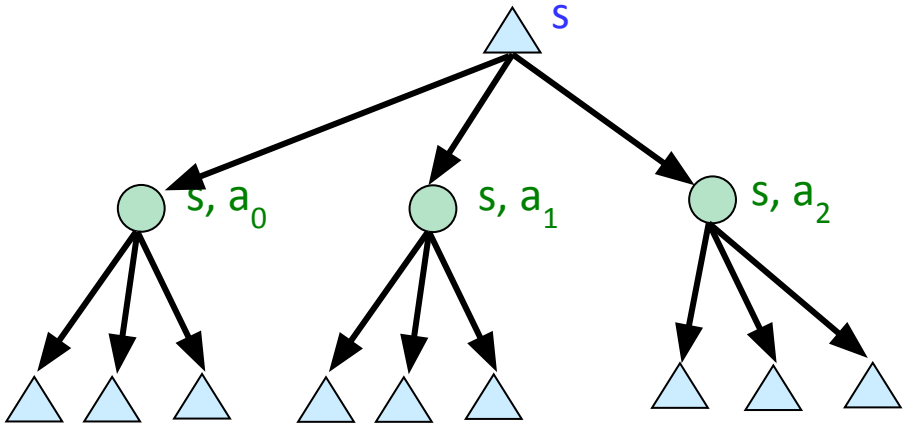
- Example of Neuroscience & AI informing each other



[A Neural Substrate of Prediction and Reward. Schultz, Dayan, Montague. 1997]

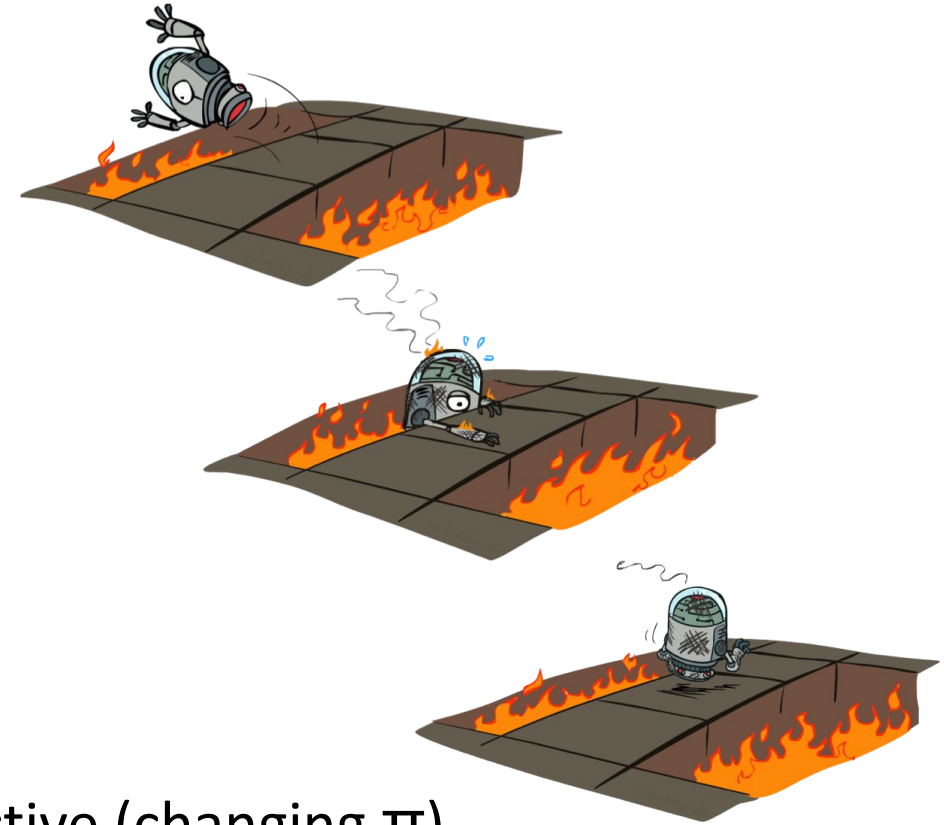
TD Value Learning

- Model-free policy evaluation! 🎉
- Bellman updates with running sample mean! 🎉



- BUT: This doesn't let us improve the policy! 😱

Active Reinforcement Learning



Active (changing π)

Approaches to reinforcement learning

There are hundreds of approaches. We will learn basic approaches for each of the following families:

1. **Model-based**: Learn the model, solve it, execute the solution
2. **Model-free**: Learn values from experiences, use to make decisions
   - Direct evaluation
   - Temporal difference learning
   - Q-learning

Passive RL

Passive RL

Passive RL

Active RL

Recall: 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:

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

Q-learning as approximate Q-iteration

- Recall the definition of Q values:
  - $Q^*(s,a)$ = expected return from doing a in s and then behaving optimally thereafter; and $\pi^*(s) = \text{argmax}_a Q^*(s,a)$

- Bellman equation for Q values:
  - $Q^*(s,a) = \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma \max_{a'} Q^*(s',a')]$
- Approximate Bellman update for Q values:
  - $Q(s,a) \leftarrow (1-\alpha) \cdot Q(s,a) + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} Q(s',a')]$

Q target

– We obtain, iteratively, a policy from learned Q(s,a), with no model!
  - (No free lunch: Q(s,a) table is |A| times bigger than V(s) table)

TD Q-Learning

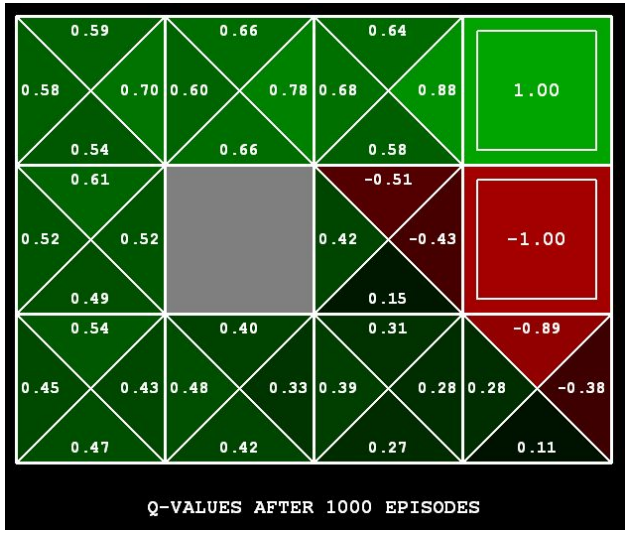
Learn Q(s,a) values as you go

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

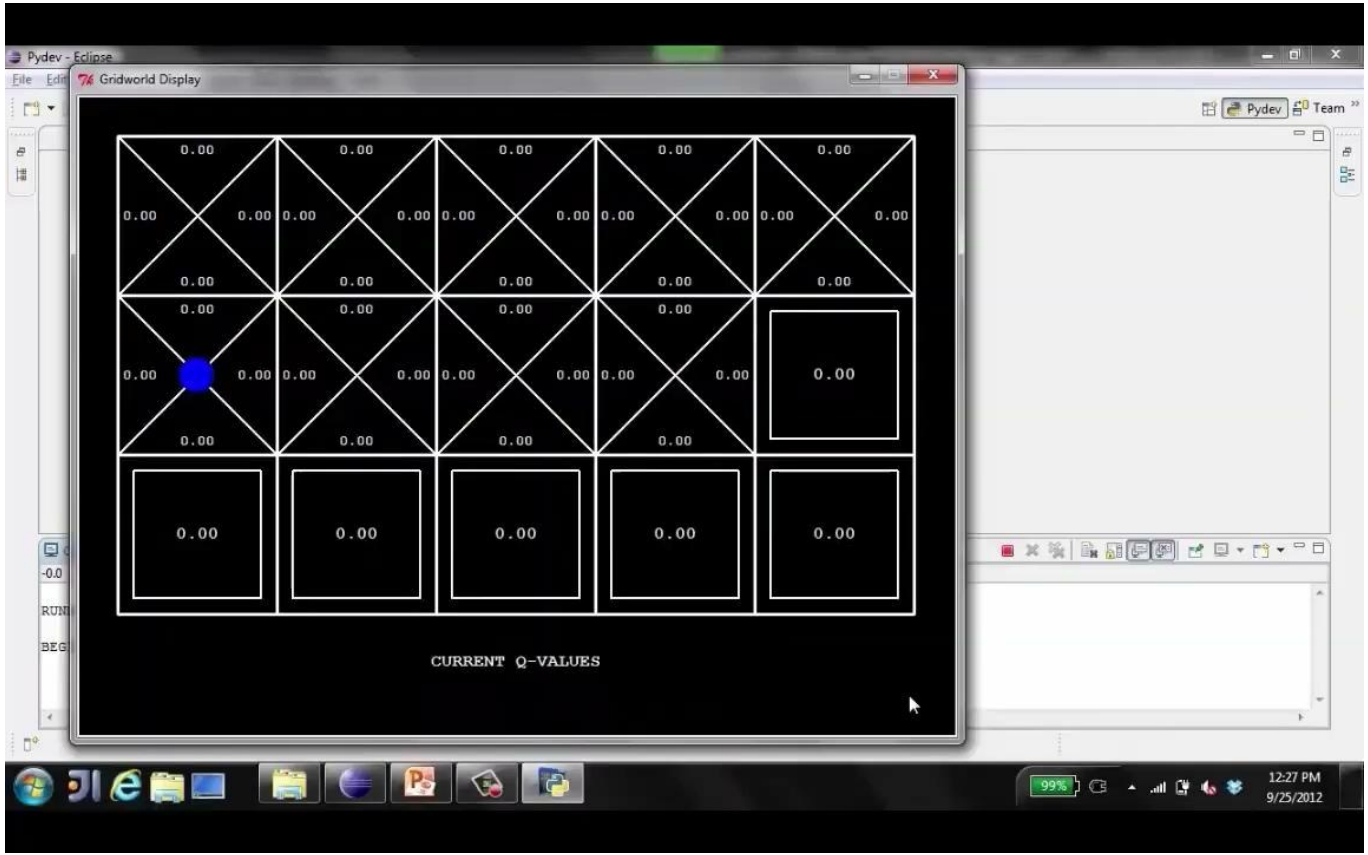
$$q\_target = R(s,a,s') + \gamma \max_{a'} Q(s',a')$$

- Incorporate the new estimate into a running average:

$$Q(s,a) \leftarrow (1-\alpha)\, Q(s,a) + \alpha \cdot [q\_target]$$



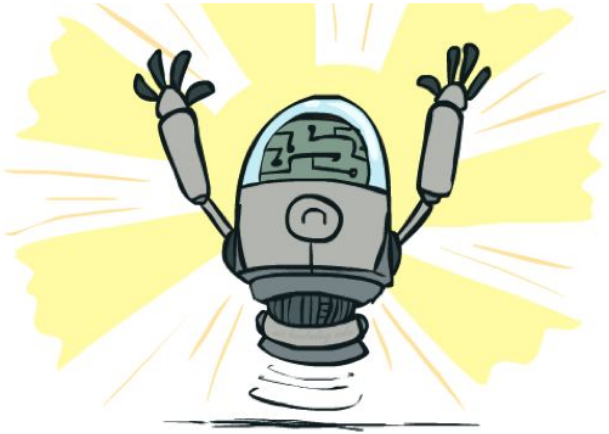
Video of Demo Q-Learning -- Gridworld



TD Q-Learning Properties

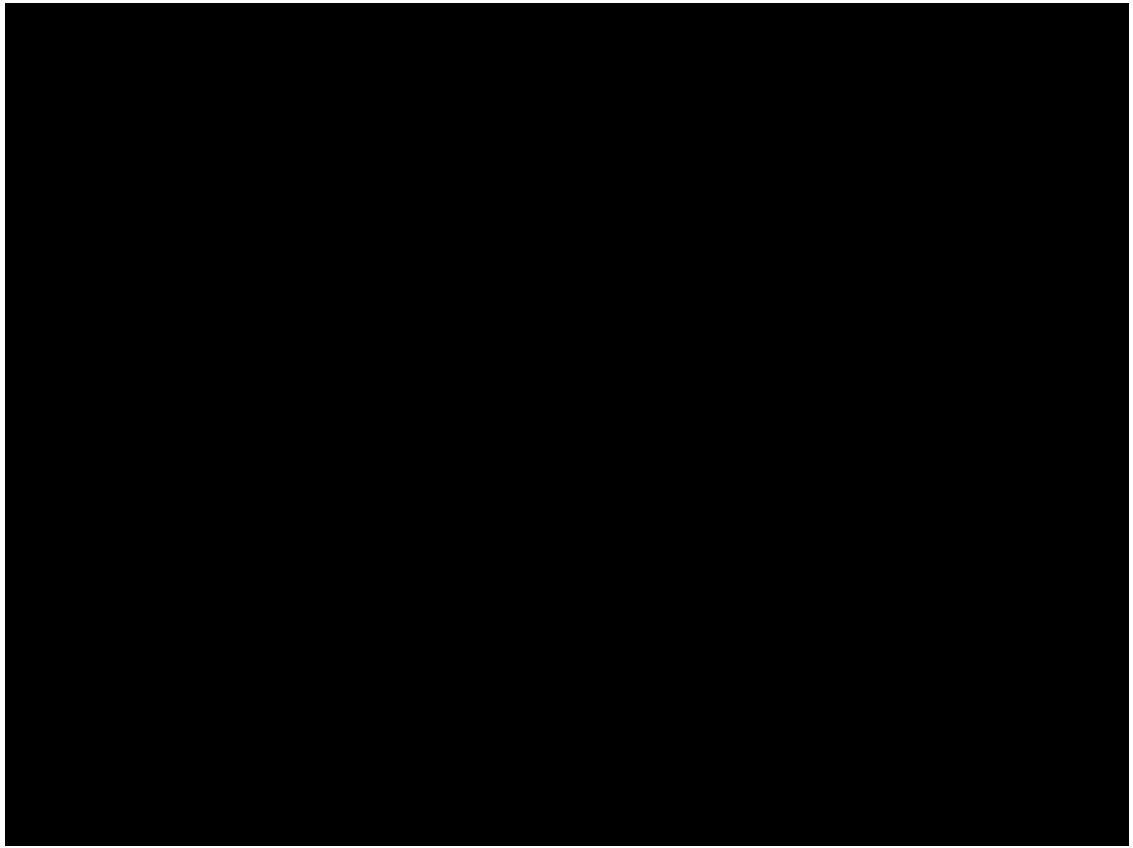
- Amazing result: Q-learning converges to optimal policy -- even if samples are generated from a suboptimal policy!

- 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 -- Crawler



Example: AlphaGo (2016)

Summary

- RL solves MDPs via direct experience of transitions and rewards
- There are several approaches:
  - Learn the MDP model and solve it
  - Learn V directly from sums of rewards, or by TD local adjustments
    - Still need a model to make decisions by lookahead
  - Learn Q by local Q-learning adjustments, use it directly to pick actions
  - (and about 100 other variations)
- Big missing pieces:
  - How to explore without too much regret?
  - How to scale this up to Tetris (10^{60}), Go (10^{172}), StarCraft ($|A|=10^{26}$)?