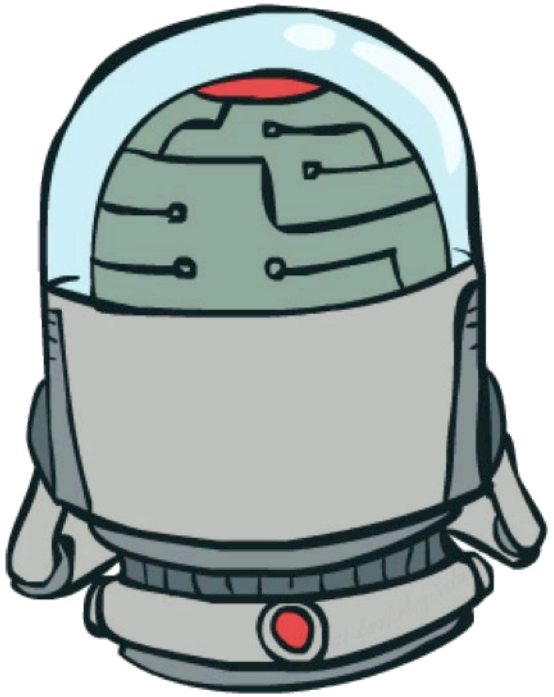
# CS7IS2: Artificial Intelligence

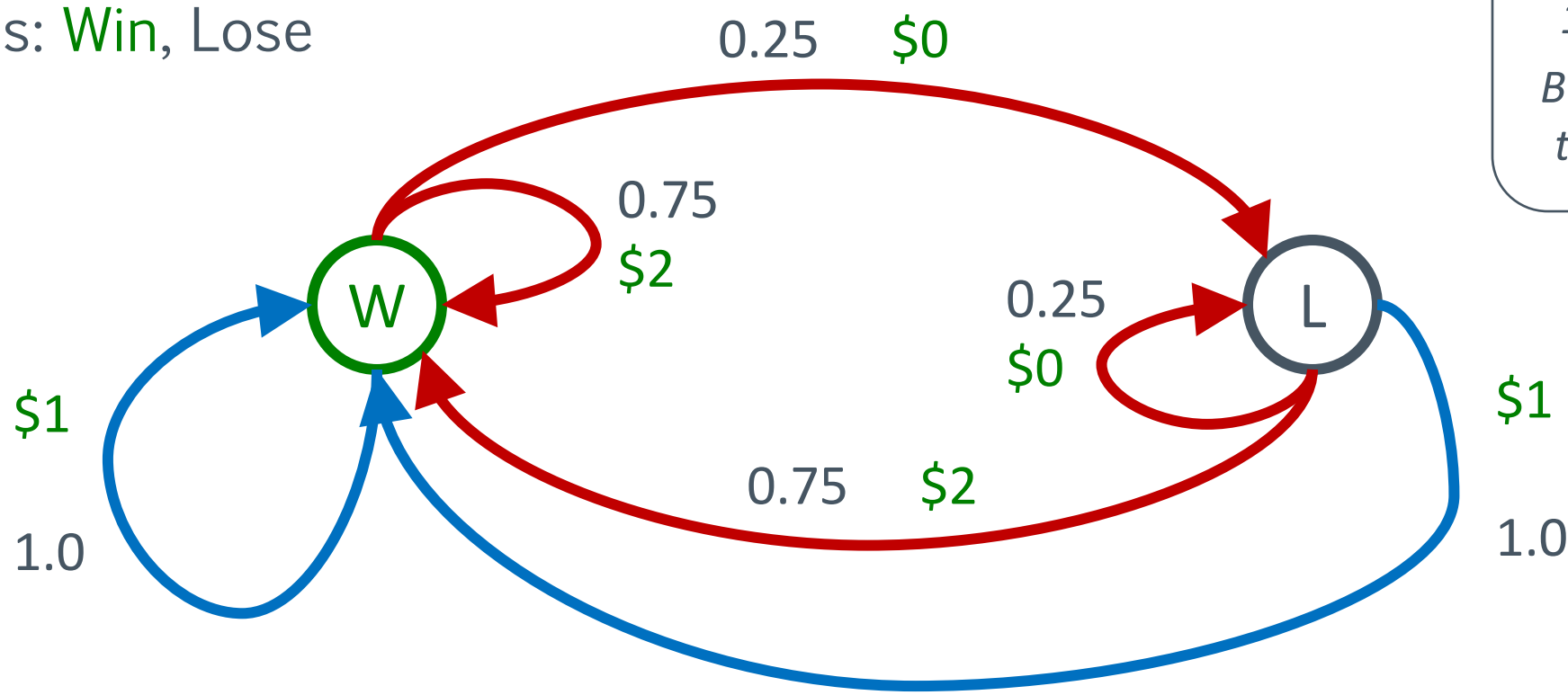
## Reinforcement Learning

# Double Bandits

# Double-Bandit MDP

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




*No discount*

*100 time steps*

*Both states have the same value*

# Offline Planning

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

*No discount*
*100 time steps*
*Both states have the same value*

| | Value |
|---|---|
| Play Red | 150 |
| Play Blue | 100 |

# Let's Play!

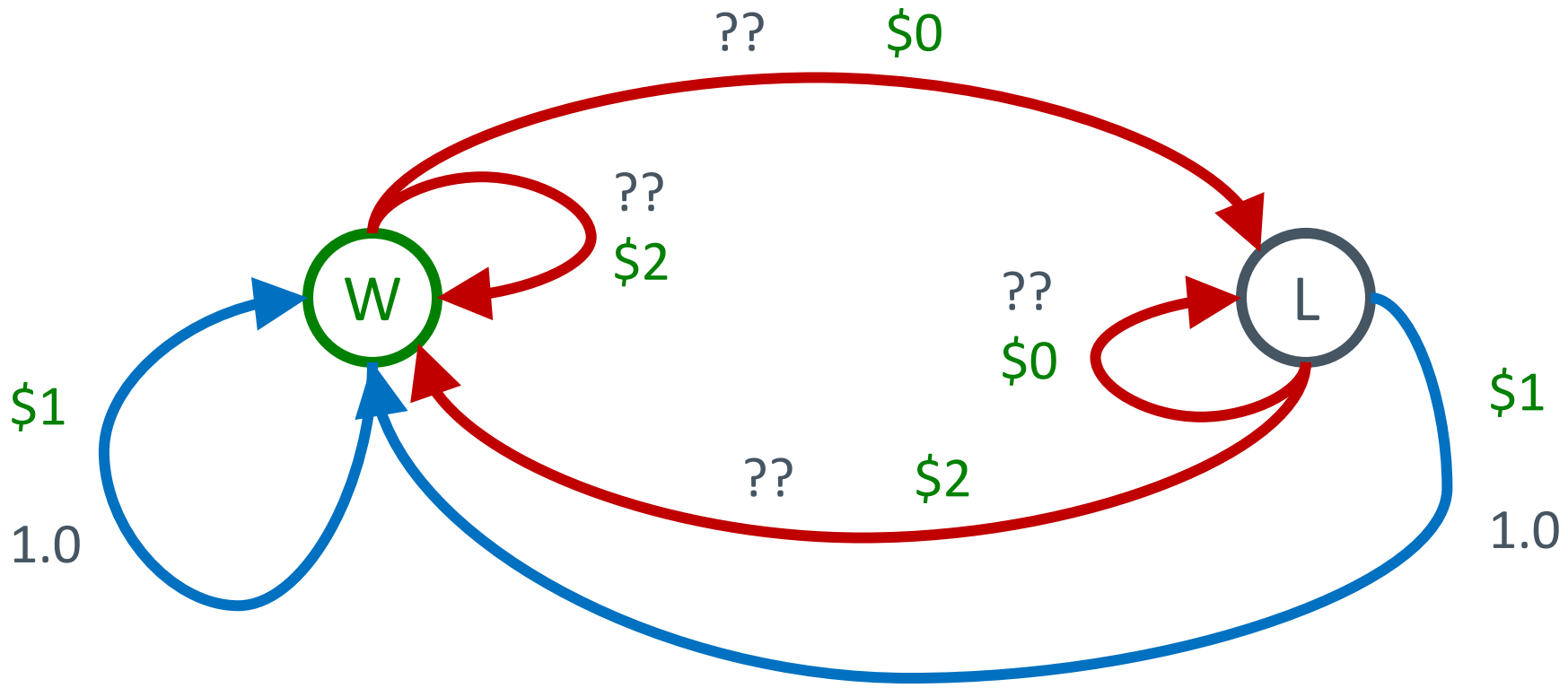


$2 $2 $0 $2 $2

$2 $2 $0 $0 $0

# Online Planning

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



# Let's Play!



$0 $0 $0 $2 $0

$2 $0 $0 $0 $0

# What Just Happened?

- That wasn't planning, it was learning!
  - Specifically, reinforcement learning
  - There was an MDP, but you couldn't solve it with just computation
  - You needed to actually act to figure it out

- Important ideas in reinforcement learning that came up
  - Exploration: you have to try unknown actions to get information
  - Exploitation: eventually, you have to use what you know
  - Regret: even if you learn intelligently, you make mistakes
  - Sampling: because of chance, you have to try things repeatedly
  - Difficulty: learning can be much harder than solving a known MDP

# Reinforcement Learning

# Background and literature

- Reinforcement Learning: An Introduction, by Richard S. Sutton and Andrew G. Barto

http://www.incompleteideas.net/book/RLbook2020.pdf

- Inspiration from psychology and neuroscience

http://www.princeton.edu/~yael/ICMLTutorial.pdf

- Deep Reinforcement Learning overview

https://arxiv.org/abs/1810.06339

- Deep Reinforcement learning Hands-on, by Maxime Lapan (second edition 2020)
  - https://github.com/PacktPublishing/Deep-Reinforcement-Learning-Hands-On-Second-Edition all the code from the book

# Unsupervised vs supervised learning

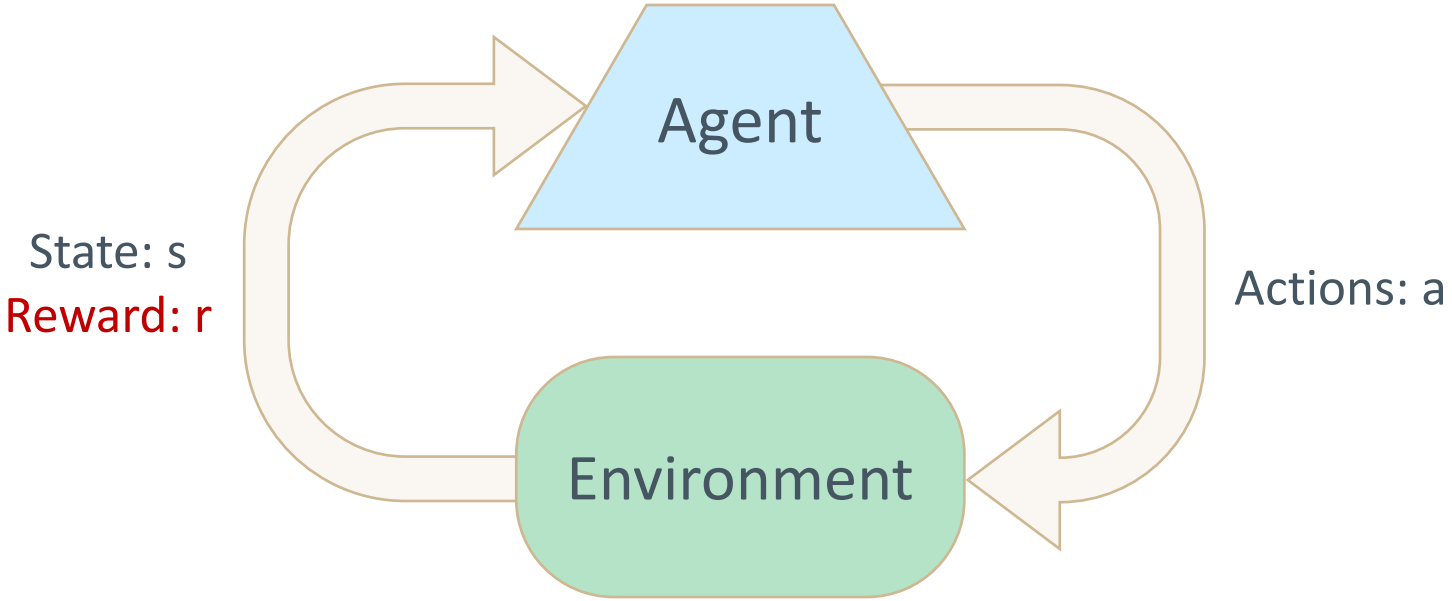
- Supervised learning
  - learning from a training set of labelled examples provided by a knowledgeable external supervisor
  - each example is a description of a situation together with a specification (the label) of the correct action the system should take to that situation (eg category to which it belongs to)
  - Extrapolate/generalize to situations not present in the training set
  - Eg classification, regression
- Unsupervised learning
  - finding structure/patterns hidden in collections of unlabeled data
  - Eg clustering, self-organizing maps

# Reinforcement learning

- Third learning paradigm
  - it does not rely on examples of correct behaviour (trial-and-error learning instead)
  - But it is reinforcement learning is trying to maximize a reward signal instead of trying to find hidden structure

- Tabular representation vs approximation functions
  - Deep RL - reinforcement learning with function approximation by deep neural networks.

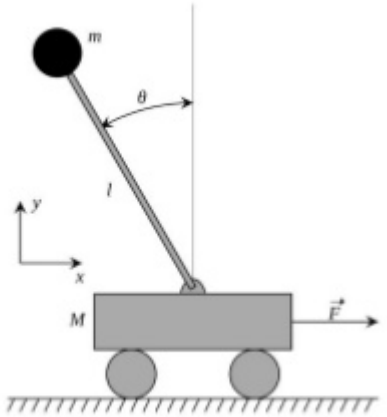
# Reinforcement Learning



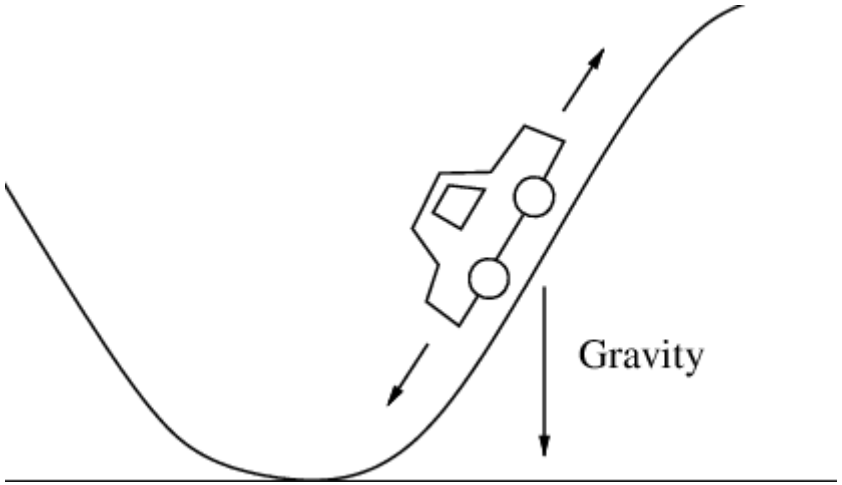

- Basic idea:
  - Receive feedback in the form of rewards
  - Agent's utility is defined by the reward function
  - Must (learn to) act so as to maximize expected rewards
  - All learning is based on observed samples of outcomes!

# Examples – Cart pole

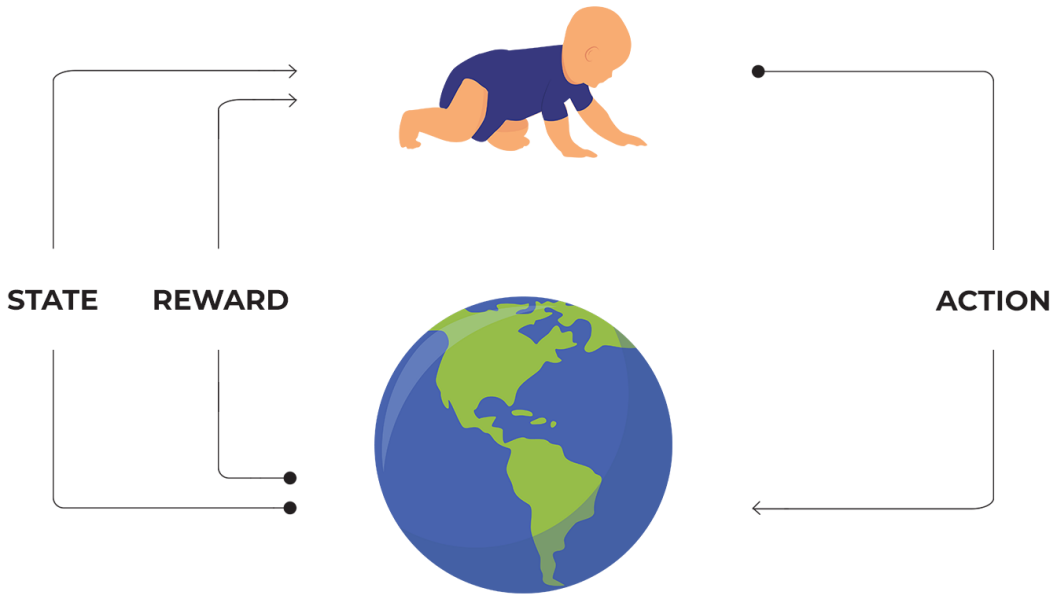
**Cart-Pole Problem**
Objective: Balance a pole on top of a movable car



# Examples – mountain car



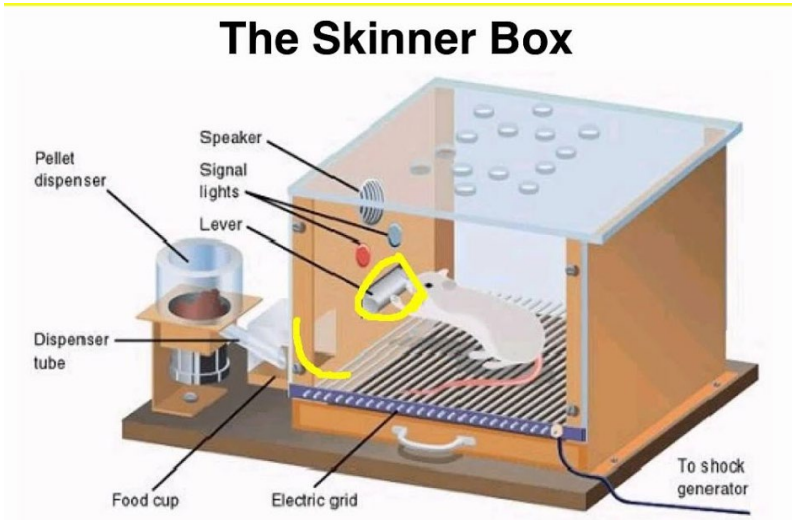
# Reinforcement Learning - Humans



# Reinforcement Learning - Animals

# Reinforcement Learning and Psychology

- **Operant conditioning** - type of associative learning process through which the strength of a behavior is modified by reinforcement or punishment
- 1938 - Skinner's "The Behavior of Organisms: An Experimental Analysis" demonstrated learning through reinforcement in animals and humans
- Applications
  - Addiction
  - Parenting
  - Animal training
  - Economics
  - Gambling
  - Military training



# Reinforcement Learning and Neuroscience


Deep Reinforcement Learning and its Neuroscientific Implications

**Matthew Botvinick**[1,2], **Jane X. Wang**[1]
**Will Dabney**[1], **Kevin J. Miller**[1,2], **Zeb Kurth-Nelson**[1,2]

[1]DeepMind, UK, [2]University College London, UK

July 9, 2020

# RL Applications - Games

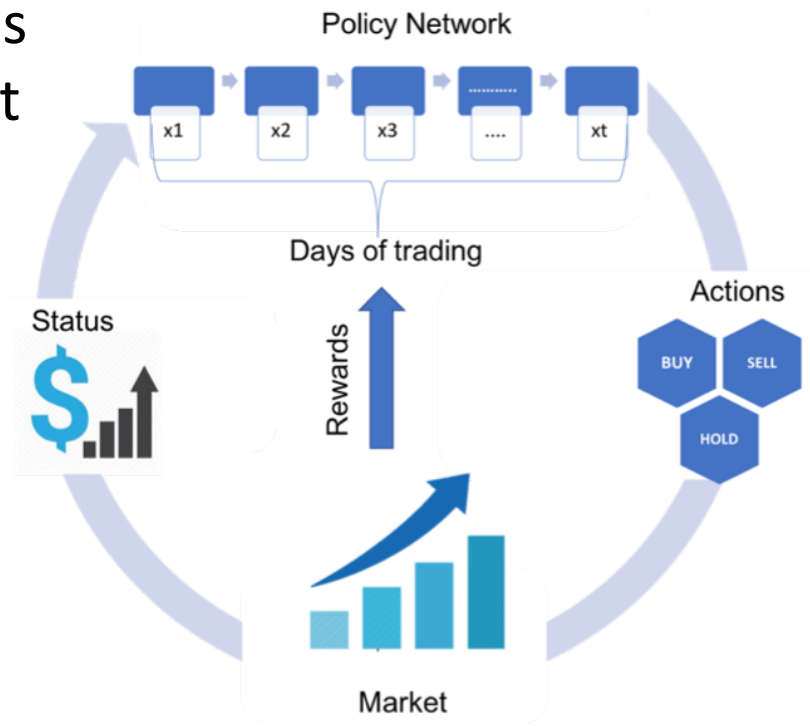
- AlphaGo
  - https://deepmind.com/research/case-studies/alphago-the-story-so-far
  - https://www.youtube.com/watch?v=WXuK6gekU1Y
- AlphaStar
  - https://deepmind.com/blog/article/AlphaStar-Grandmaster-level-in-StarCraft-II-using-multi-agent-reinforcement-learning
- Atari with DQN
  - Mnih, V., Kavukcuoglu, K., Silver, D. *et al.* Human-level control through deep reinforcement learning. *Nature* **518,** 529–533 (2015). https://doi.org/10.1038/nature14236
- MineRL
  - https://minerl.io/
  - imitation learning datasets with **over 60 million frames** of recorded human player data
  - https://www.aicrowd.com/challenges/neurips-2021-minerl-competition?s=09

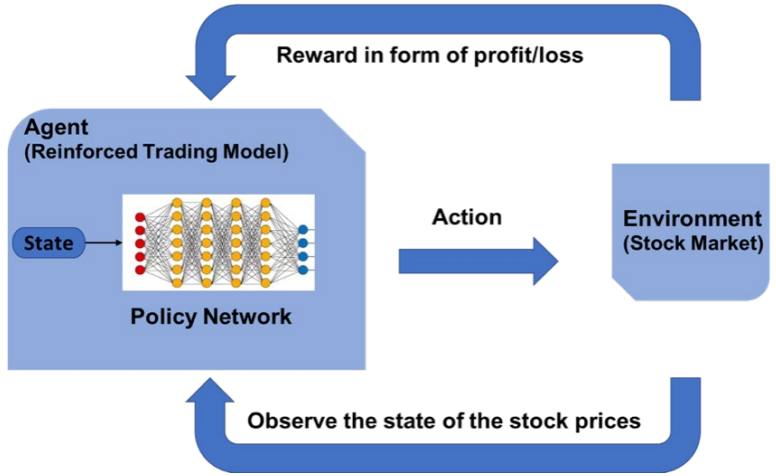
# RL Applications – Self-driving cars

- Wayve.ai – learn to drive in a day
  - https://wayve.ai/blog/learning-to-drive-in-a-day-with-reinforcement-learning/
- AWS DeepRacer
  - fully autonomous 1/18th scale race car driven b reinforcement learning
  - https://aws.amazon.com/deepracer/
- Multiple directions explored in research
  - B. R. Kiran *et al.*, "Deep Reinforcement Learning for Autonomous Driving: A Survey," in *IEEE Transactions on Intelligent Transportation Systems*, doi: 10.1109/TITS.2021.3054625.

# RL Applications – Finance

- Trading bots
- Price-setting strategies
- Portfolio management



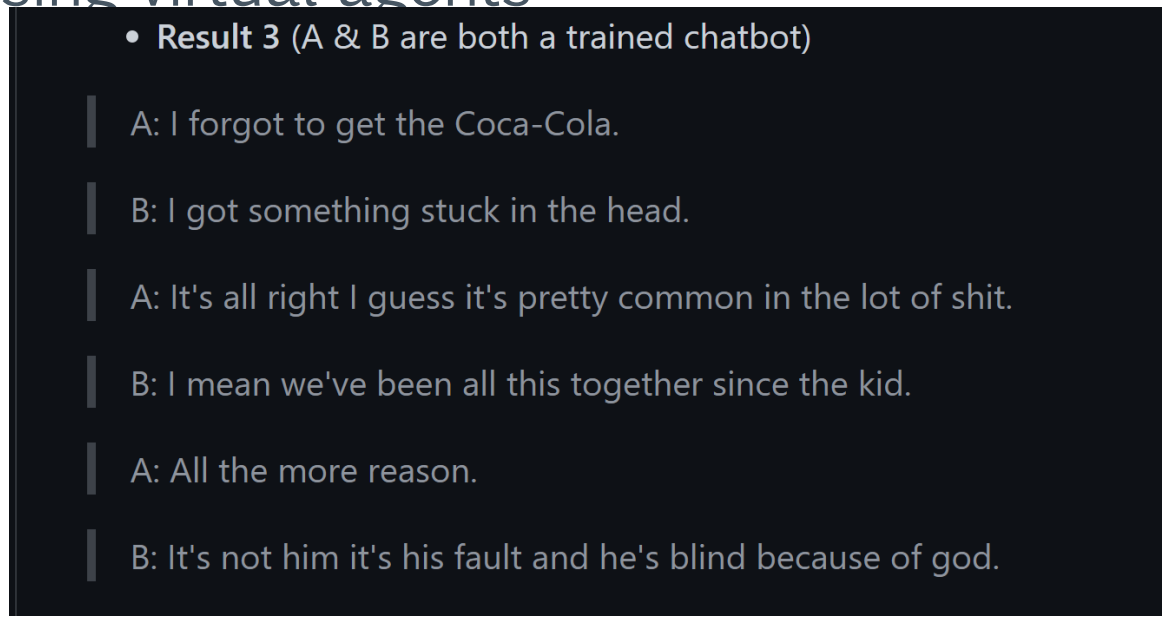




https://medium.com/ibm-data-ai/reinforcement-learning-the-business-use-case-part-2-c175740999

# RL Applications – NLP

- RL from Human Feedback
- Chatbots - dialogue generation
  - Simulate conversations using virtual agents

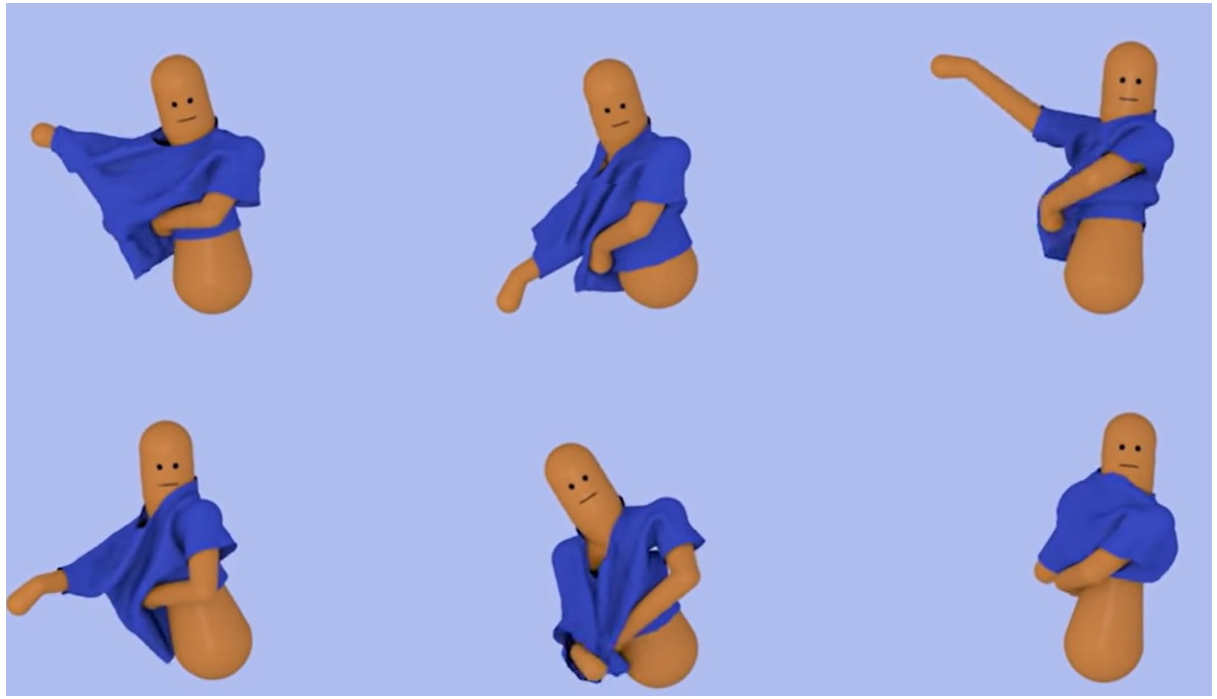
- Text summarization
- Question answering
- Machine translation




https://github.com/pochih/RL-Chatbot (2017)

# RL Applications – Robotics

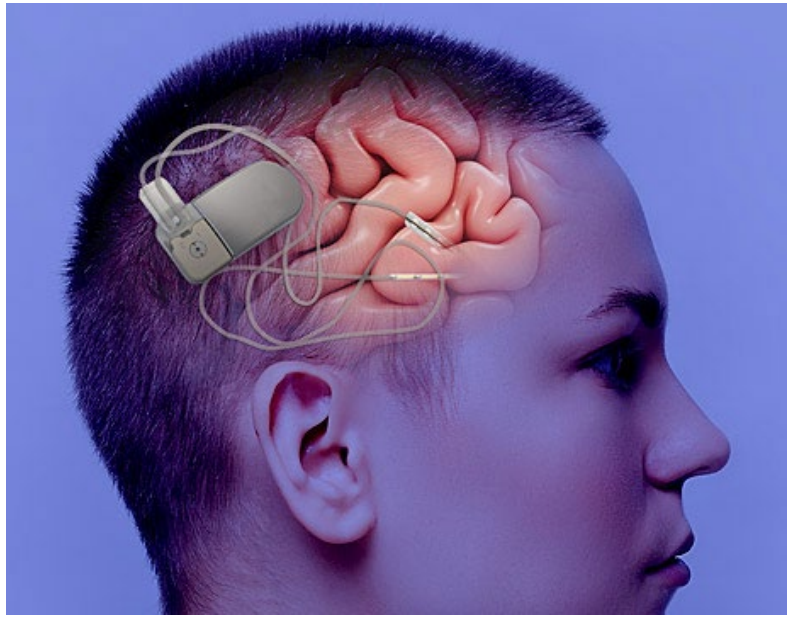
- http://rail.eecs.berkeley.edu/
- Deep RL in Robotics and Autonomous Decision Making – Sergey Levine
  - https://www.youtube.com/watch?v=lEKYi5OLABI



Learning To Dress: Synthesizing Human Dressing Motion via Deep Reinforcement Learning

# RL Applications – Healthcare

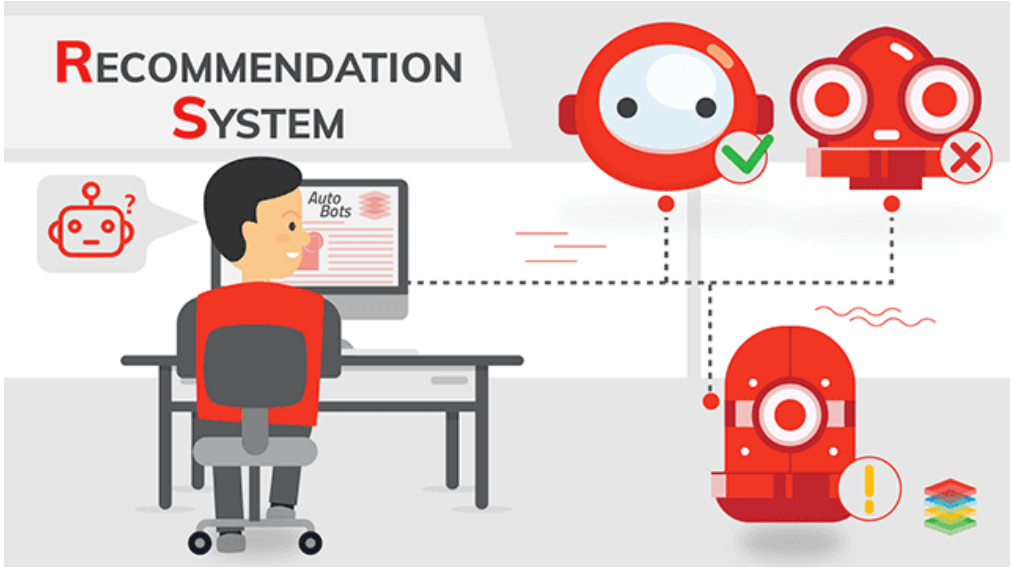
- Reinforcement Learning in Healthcare: A Survey
  - https://ui.adsabs.harvard.edu/abs/2019arXiv190808796Y/abstract
  - treatment recommendation, automatically learning an optimal neurostimulation strategy for the treatment of epilepsy



# RL Applications – Recommender systems

- News recommendation
- https://reagent.ai/ - open source end-to-end platform for applied reinforcement learning (RL) developed and used at Facebook.

- **Reinforcement learning based recommender systems: A survey.** M. Mehdi Afsar, Trafford Crump, Behrouz Far
  - https://arxiv.org/abs/2101.06286



# RL Applications – Cluster resource management, energy use


https://deepmind.com/blog/article/deepmind-ai-reduces-google-data-centre-cooling-bill-40

# RL Applications – Design/Engineering

- TPU design
  - design the next generation of Google's artificial intelligence (AI) accelerators, and has the potential to save thousands of hours of human effort for each new generation.
  - https://www.nature.com/articles/s41586-021-03544-w

- McKinsey Analytics - test new hydrofoil designs by sailing them on Emirates Team New Zealand's simulator
  - Teach an AI bot how to become a professional sailor, 1000s bots running in parallel learning from each other

# RL Applications – Smart Cities

**A Survey on Reinforcement Learning Models and Algorithms for Traffic Signal Control**

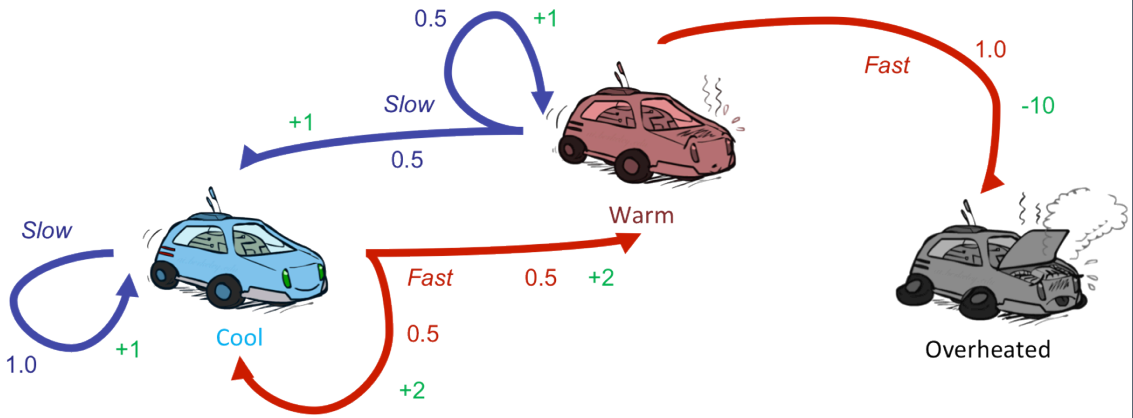
https://dl.acm.org/doi/10.1145/3068287

Reinforcement learning for demand response: A review of algorithms and modeling techniques

https://ideas.repec.org/a/eee/appene/v235y2019icp1072-1089.html

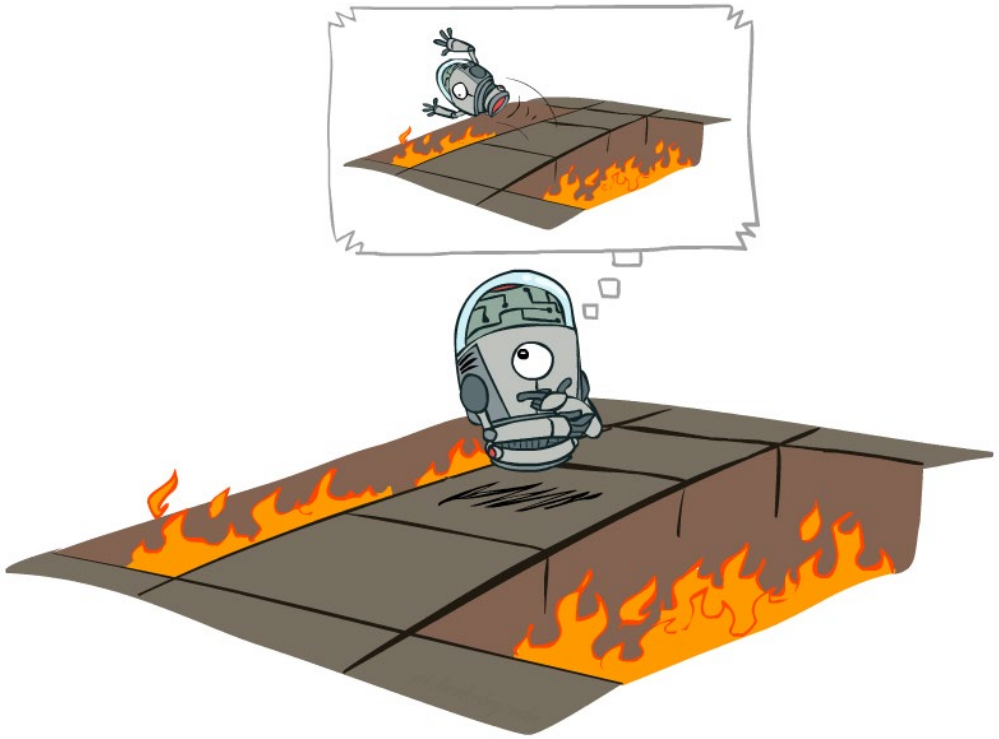
# RL

- Still assume a Markov decision process (MDP):
  - A set of states $s \in S$
  - A set of actions (per state) A
  - A model T(s,a,s')
  - A reward function R(s,a,s')
- Still looking for a policy $\pi(s)$

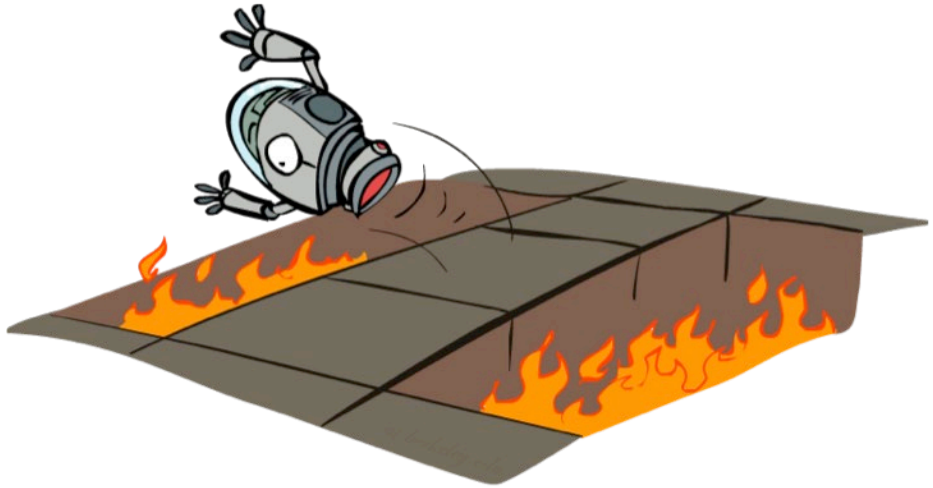



- New twist: don't know T or R
  - I.e. we don't have the model of the environment
  - we don't know which states are good or what the actions do
  - Must actually try out actions and states to learn

# Offline (MDPs) vs.  Online (RL)



Offline Solution



Online Learning

# Model-based learning

# Model-based learning

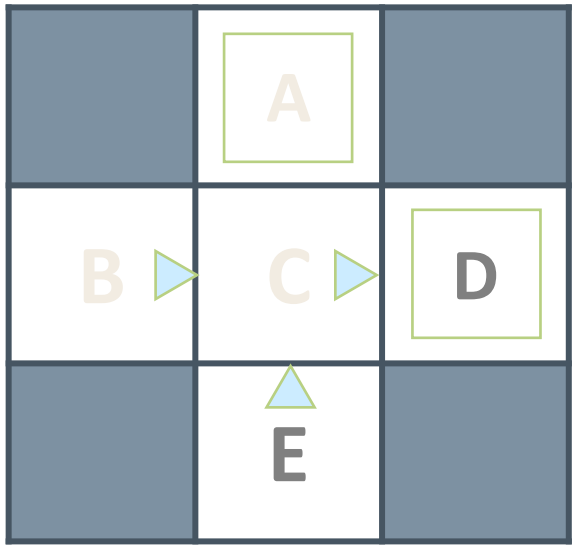
- Model-Based Idea:
  - Learn an approximate model based on experiences
  - Solve for values as if the learned model were correct

- 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

# Model-based example

## Input Policy $\pi$




*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

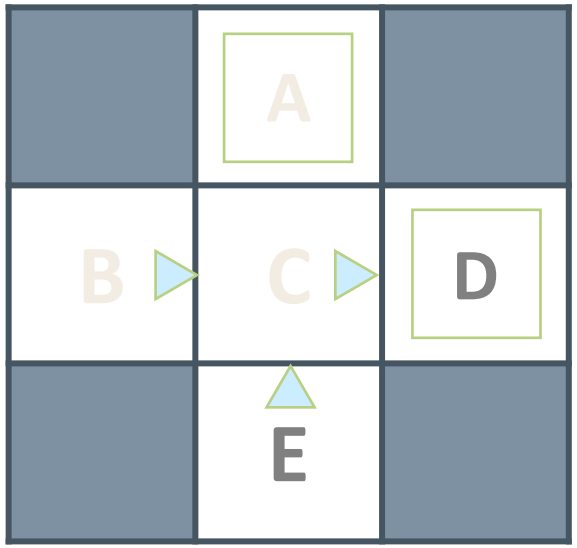
## Learned Model

$\hat{T}(s, a, s')$

$\hat{R}(s, a, s')$

# Model-based example

## Input Policy $\pi$




*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 RL

# Model-free RL

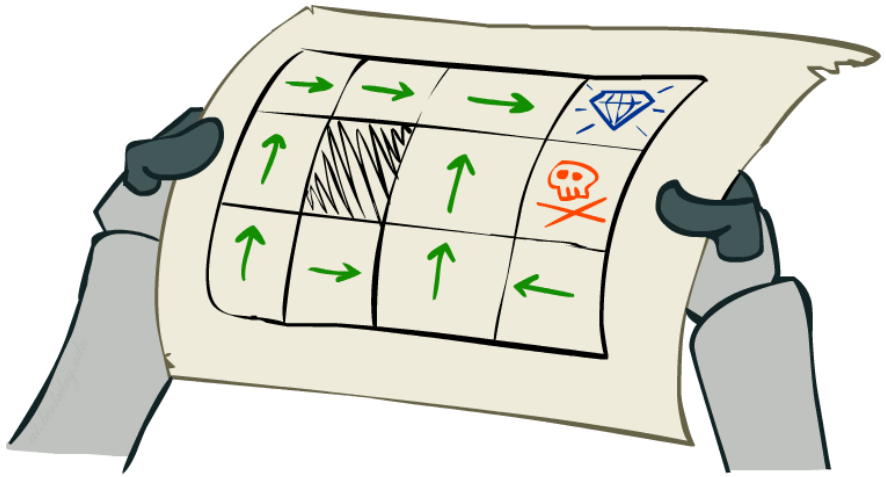
- Passive vs active RL
- Passive – estimate values of states
- Active – how do we collect the data to estimate values (ie taking actions)

- Passive – direct and indirect evaluation

# Passive RL

# Passive RL

- 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



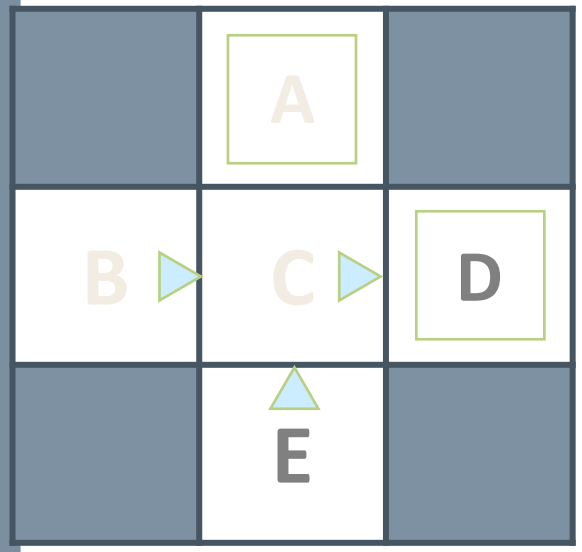
- In this case:
  - Learner is "along for the ride"
  - No choice about what actions to take
  - Just execute the policy and learn from experience
  - This is NOT offline planning! You actually take actions in the world.

# Direct evaluation

- Goal: Compute values for each state under $\pi$
- Idea: Average together observed sample values
  - Act according to $\pi$
  - 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

# Direct evaluation

## Input Policy $\pi$




*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

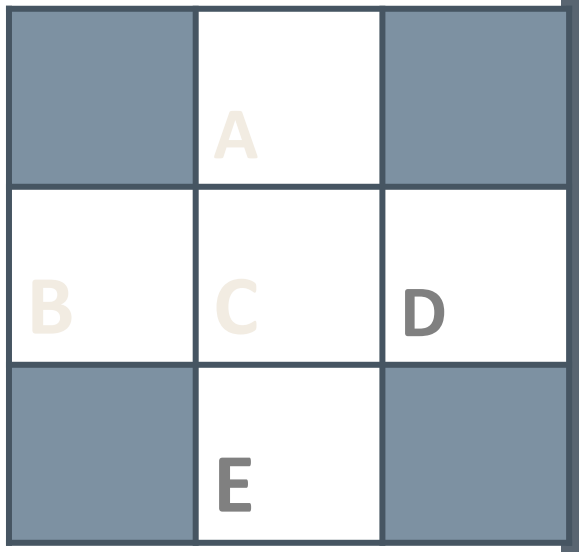
### Episode 3

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

### Episode 4

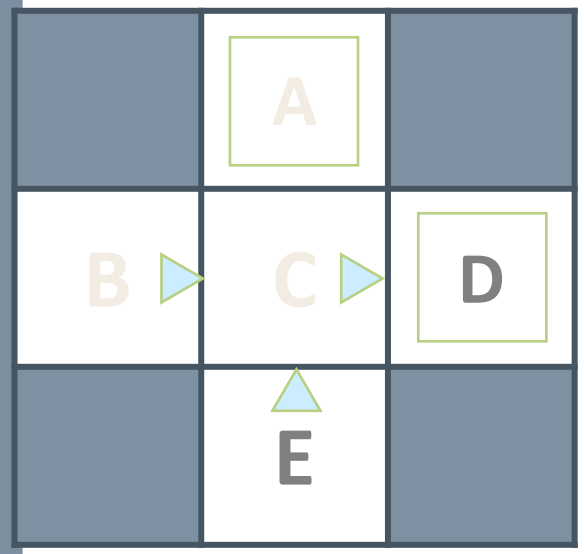
E, north, C, -1
C, east, A, -1
A, exit, x, -10

## Output Values



# Direct evaluation

## Input Policy $\pi$




*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

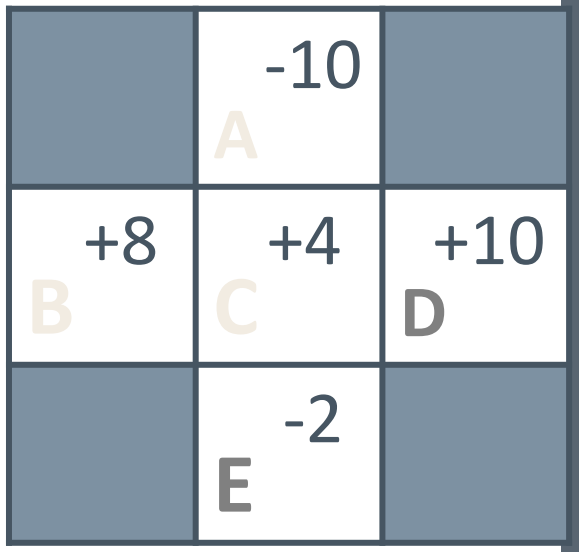
### Episode 3

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

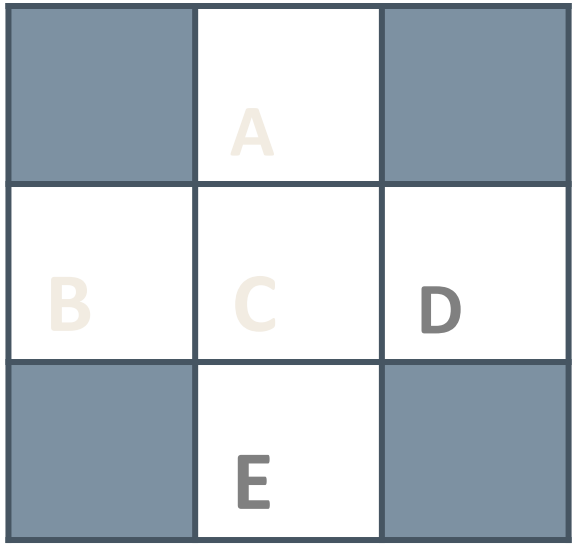


# Problems with direct evaluation

- 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



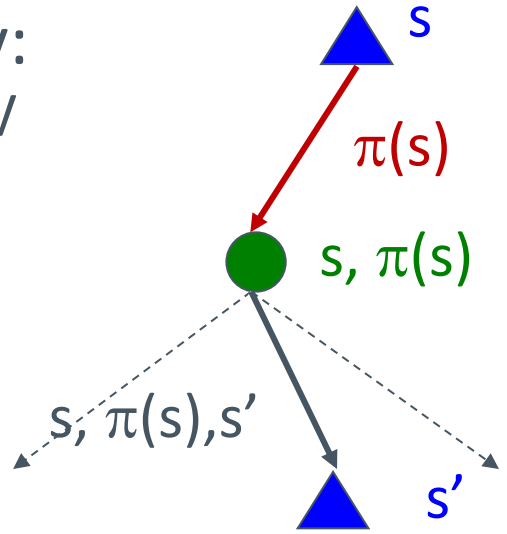

*If B and E both go to C under this policy, how can their values be different?*

# Why Not Use Policy Evaluation?

- Simplified Bellman updates calculate V for a fixed policy:
  - Each round, replace V with a one-step-look-ahead layer over V

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

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




  - This approach fully exploited the connections between the states
  - Unfortunately, we need T and R to do it!

- Key question: how can we do this update to V without knowing T and R?
  - In other words, how 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:

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

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

$$sample_1 = R(s, \pi(s), s'_1) + \gamma V^{\pi}_{k}(s'_1)$$

$$sample_2 = R(s, \pi(s), s'_2) + \gamma V^{\pi}_{k}(s'_2)$$

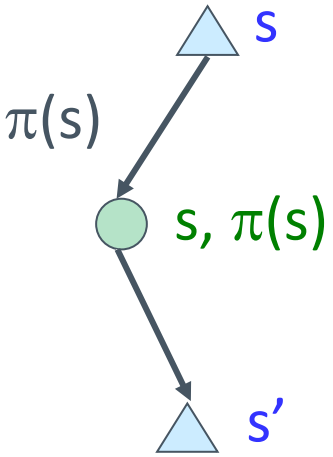
$$\cdots$$

$$sample_n = R(s, \pi(s), s'_n) + \gamma V^{\pi}_{k}(s'_n)$$

$$V^{\pi}_{k+1}(s) \leftarrow \frac{1}{n} \sum_{i} sample_i$$

# Temporal difference learning

- Big idea: learn from every experience!
  - Update V(s) each time we experience a transition (s, a, s', r)
  - Likely outcomes s' will contribute updates more often




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

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

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

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

# 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

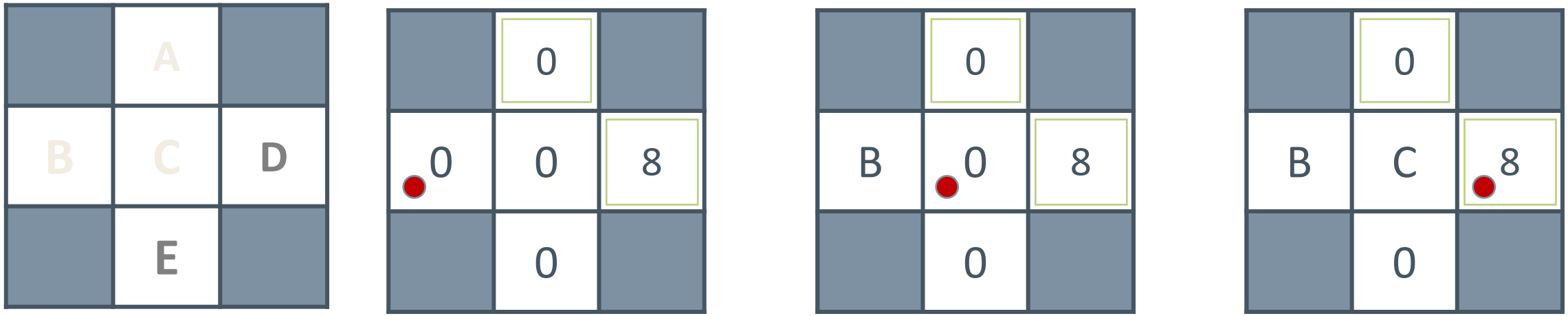
# TD learning example

States

Observed Transitions

B, east, C, -2

C, east, D, -2




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

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

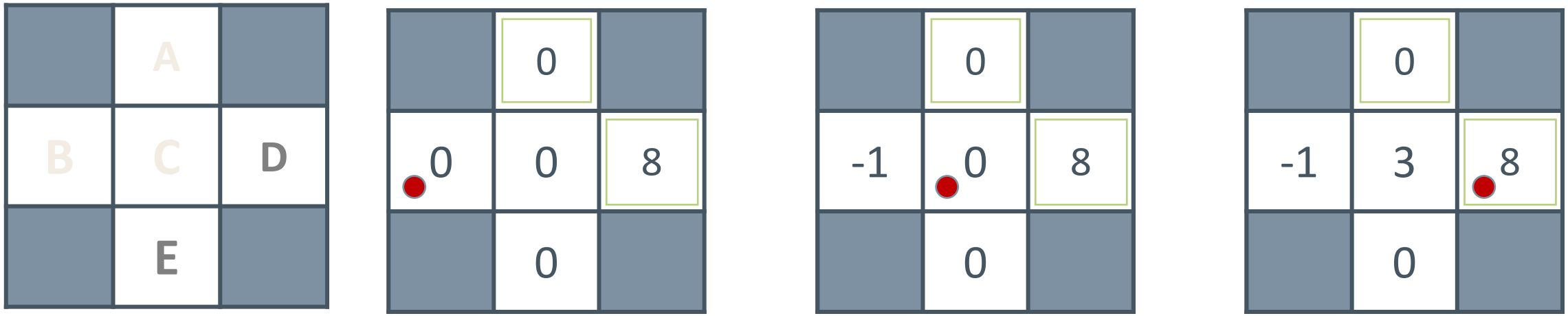
# TD learning example

States

Observed Transitions

B, east, C, -2

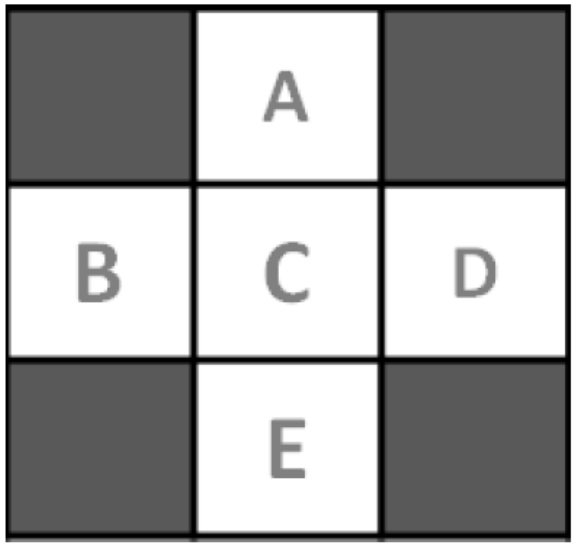
C, east, D, -2




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

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

# TD exercise




Suppose that we have the following observed transitions:
(B, East, C, 2), (C, South, E, 4), (C, East, A, 6), (B, East, C, 2)

The initial value of each state is 0. Assume that $\gamma = 1$ and $\alpha = 0.5$.

1. What are the learned values from TD learning after all four observations?

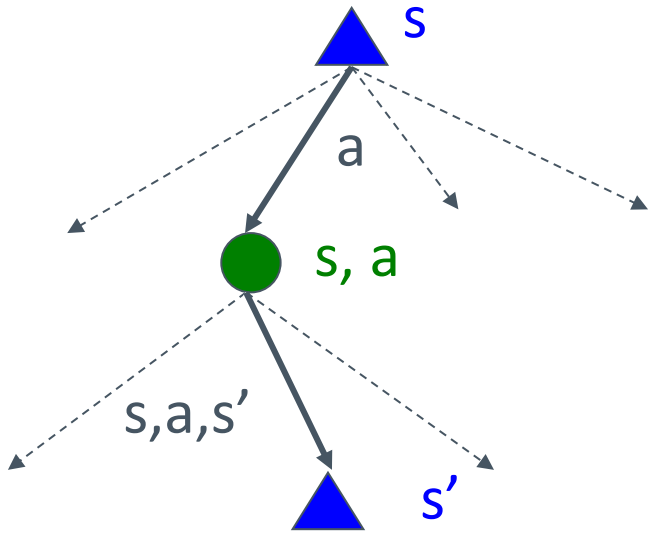
$$V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + \alpha\left[R(s,\pi(s),s') + \gamma V^{\pi}(s')\right]$$

# Problems with TD learning

- TD value leaning is a model-free way to do policy evaluation, mimicking Bellman updates with running sample averages
- However, if we want to turn values into a (new) policy, we're sunk:

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

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




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

# Active RL

# 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

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

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

› Learn Q(s,a) values as you go

– Receive a sample (s,a,s',r)

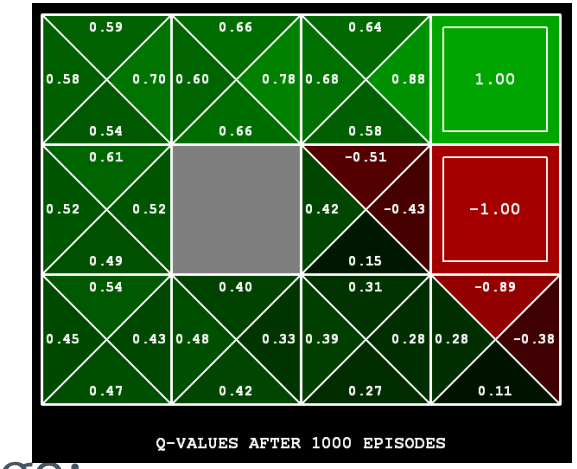
– Consider your old estimate: $Q(s,a)$

– Consider your new sample estimate:

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

– Incorporate the new estimate into a running average:

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



# 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 (!)

# Exercise

A, B, C, D, E

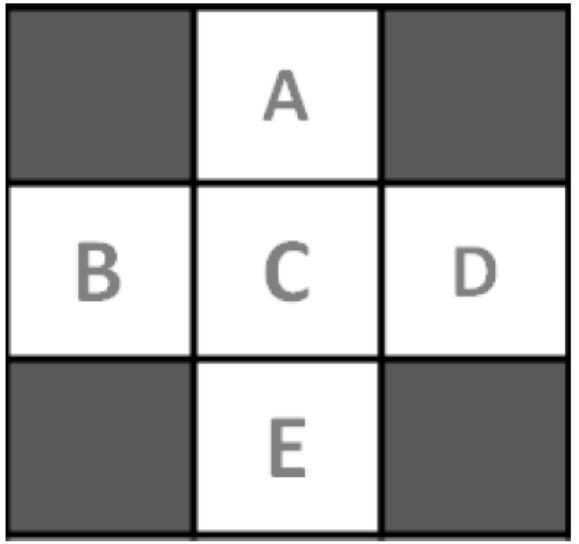
↑ ↓ ← →

$Q(B, \rightarrow) = 0.5(0) + 0.5(2 + 1(0)) = 1$

$Q(C, \rightarrow) = 0.5(0) + 0.5(4 + 1(0)) = 2$

⋮

$Q(B, \rightarrow) = 0.5(2) + 0.5(2 + (1)(2)) = 3$




↑ ↓ ← →

A
B
C
D
E

x3

2

Suppose that we have the following observed transitions:
(B, East, C, 2), (C, South, E, 4), (C, East, A, 6), (B, East, C, 2)

The initial value of each state is 0. Assume that $\gamma = 1$ and $\alpha = 0.5$.

2. What are the learned Q-values from Q-learning after all four observations?

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

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

# Exploration vs exploitation

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

# 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!

# Regret

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

# On-policy learning

- Remember Q-learning is off-policy learning
  - updating based on the best action from the next state

- On-policy learning
  - update based on the action your current policy actually takes from the next state
  - SARSA (**S**tate-Action-Reward-State-Action)

# SARSA vs Q-learning

$$Q(s,a) = \alpha\left[R(s) + \gamma Q(s',a')\right] + (1-\alpha)Q(s,a)$$

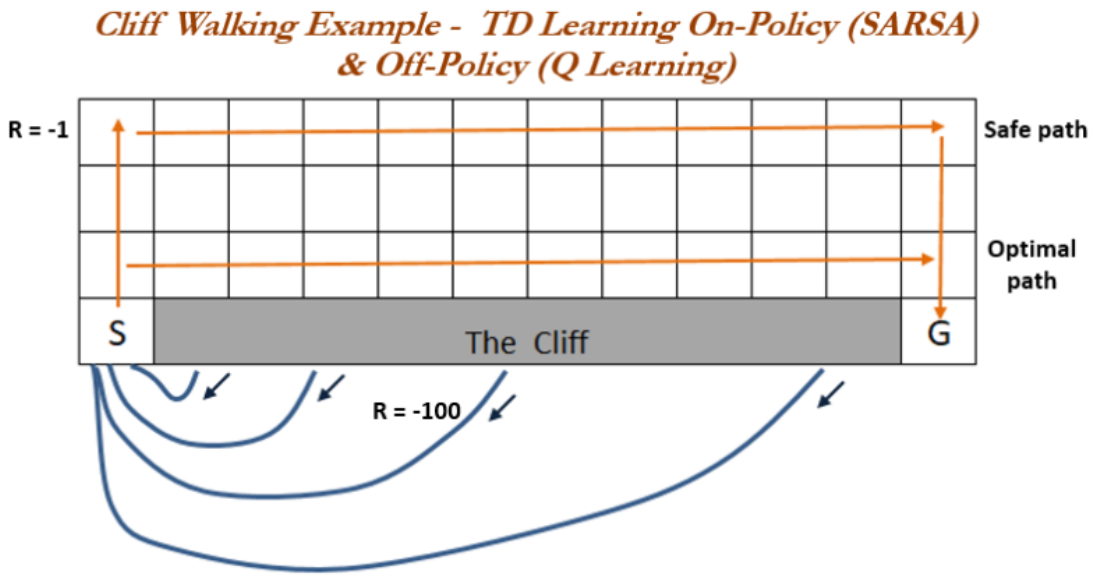
$$Q(s,a) \mathrel{+}= \alpha\left[R(s) + \gamma Q(s',a') - Q(s,a)\right]$$

Q-learning

$\gamma \max_{a'} Q(s',a')$

# SARSA vs Q-learning

› https://towardsdatascience.com/reinforcement-learning-cliff-walking-implementation-e40ce98418d4



# Approximate Q-learning

# The Story So Far: MDPs and RL

## Known MDP: Offline Solution

| Goal | Technique |
| --- | --- |
| Compute V*, Q*, $\pi$* | Value / policy iteration |
| Evaluate a fixed policy $\pi$ | Policy evaluation |

## Unknown MDP: Model-Based

| Goal | Technique |
| --- | --- |
| Compute V*, Q*, $\pi$* | VI/PI on approx. MDP |
| Evaluate a fixed policy $\pi$ | PE on approx. MDP |

## Unknown MDP: Model-Free

| Goal | Technique |
| --- | --- |
| Compute V*, Q*, $\pi$* | Q-learning |
| Evaluate a fixed policy $\pi$ | Value Learning |

# Generalizing across the states

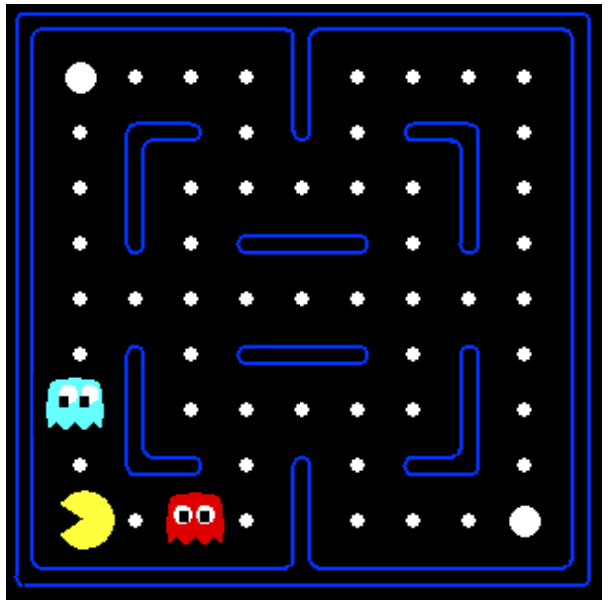
- Basic Q-Learning keeps a table of all q-values

- In realistic situations, we cannot possibly learn about every single state!
  - Too many states to visit them all in training
  - Too many states to hold the q-tables in memory

- Instead, we want to generalize:
  - Learn about some small number of training states from experience
  - Generalize that experience to new, similar situations
  - This is a fundamental idea in machine learning, and we'll see it over and over again

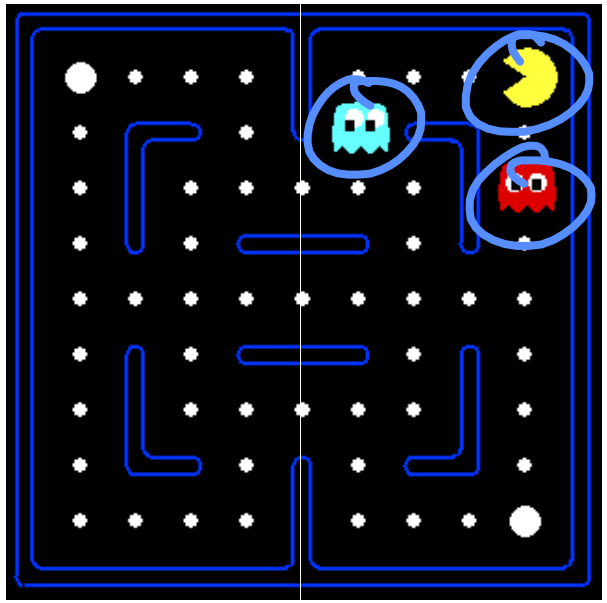
# Example: Pacman

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



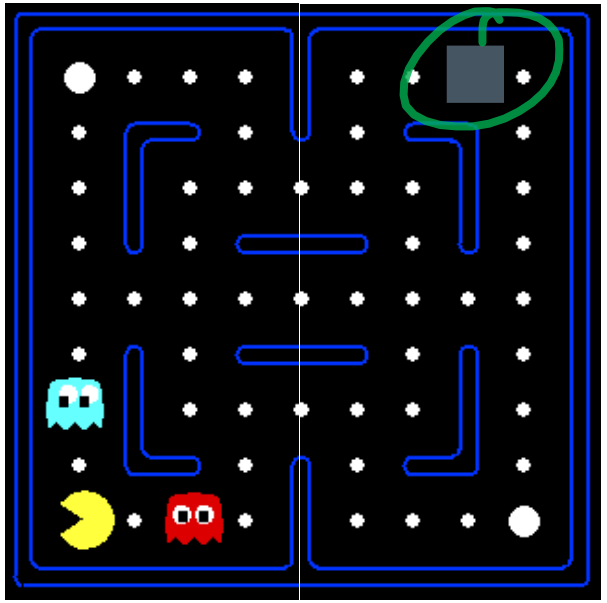
Baseline Example

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



More elements (but similar to baseline)

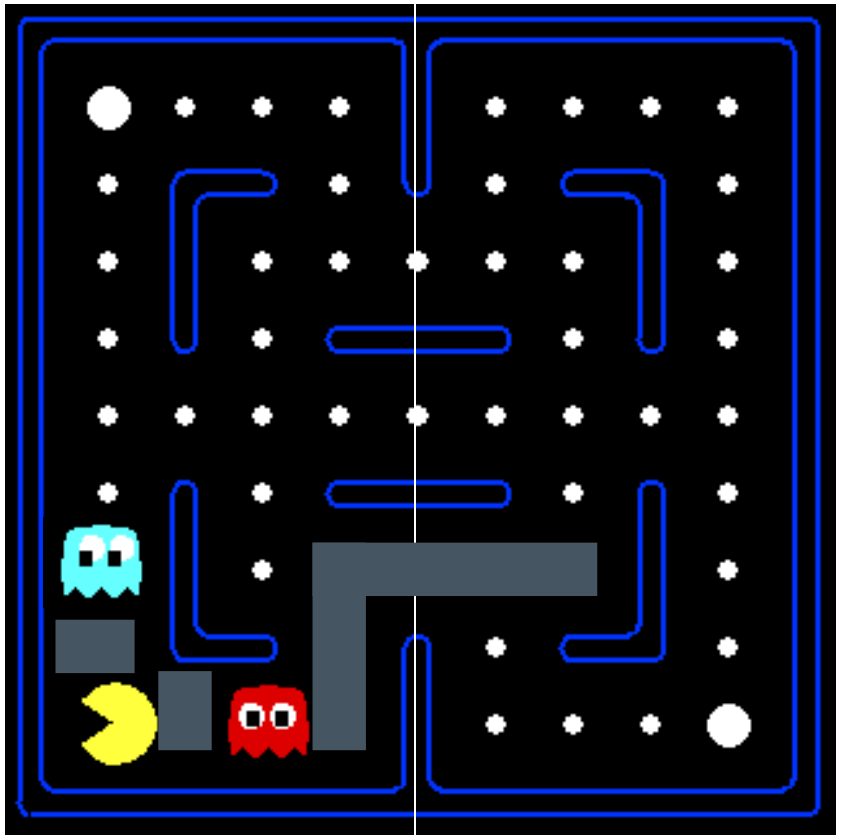
Or even this one!



One difference from baseline (Added block)

# Feature-based representation

- 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.
    - *Am I think exact state? (for a small number of important states)*
  - Can also describe a q-state (s, a) with features (e.g. action moves closer to food)



This way, different states can have similar or equal features, meaning they're actually equivalent states.

# Linear-value functions

- Using a feature representation, we can write a q function (or value function) for any state using a few weights:

$f_n$ = $n^{th}$ feature

$w_n$ = $n^{th}$ feature weight

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

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

Useful for neural nets.

- Advantage: our experience is summed up in a few powerful numbers

- Disadvantage: states may share features but actually be very different in value!

↳ Double-edged sword effect of using features.

# 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:

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

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

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

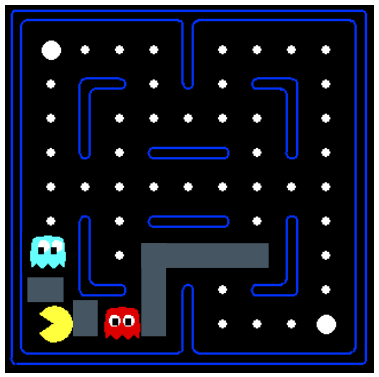
$$w_i \leftarrow w_i + \alpha\,[\text{difference}]\, f_i(s,a) \qquad \text{Approximate Q's}$$

- 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

# Example: Pacman

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

$s$

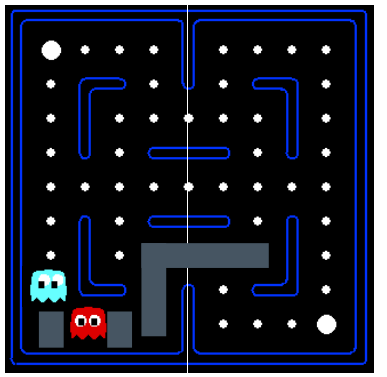


$f_{DOT}(s, \text{NORTH}) = 0.5$

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

$a = \text{NORTH}$
$r = -500$

$s'$



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

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

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

difference $= -501$

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

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

# Policy search

# Policy-search

- Problem: often the feature-based policies that work well (win games, maximize utilities) aren't the ones that approximate V / Q best
  - Q-learning's priority: get Q-values close (modeling)
  - Action selection priority: get ordering of Q-values right (prediction)
- Solution: learn policies that maximize rewards, not the values that predict them
- Policy search: start with an ok solution (e.g. Q-learning) then fine-tune by hill climbing on feature weights

# Policy-search

- Simplest policy search:
  - Start with an initial linear value function or Q-function
  - Nudge each feature weight up and down and see if your policy is better than before
- Problems:
  - How do we tell the policy got better?
  - Need to run many sample episodes!
  - If there are a lot of features, this can be impractical
- Better methods exploit lookahead structure, sample wisely, change multiple parameters…
- https://icml.cc/2015/tutorials/PolicySearch.pdf

# Policy-search

- Policy gradient search
- Actor critic
  - Combine value-based and policy-based
  - Critic - measures how good the action taken is (value-based)
  - Actor - controls how our agent behaves (policy-based)
  - A2C, A3C
  - DDPG (deep deterministic policy gradient)
  - PPO (policy proximal optimisation)

# Deep RL

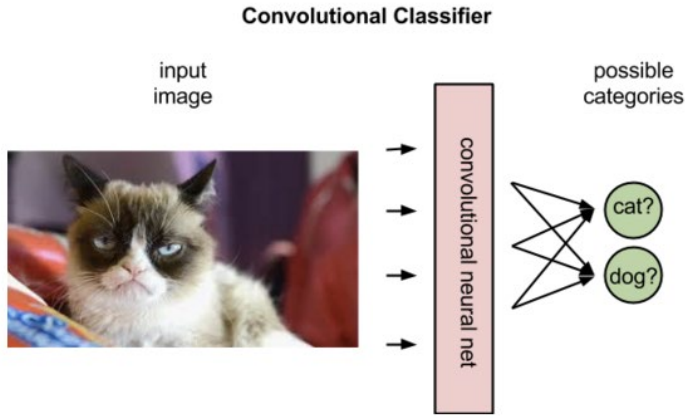
# RL+ DNN

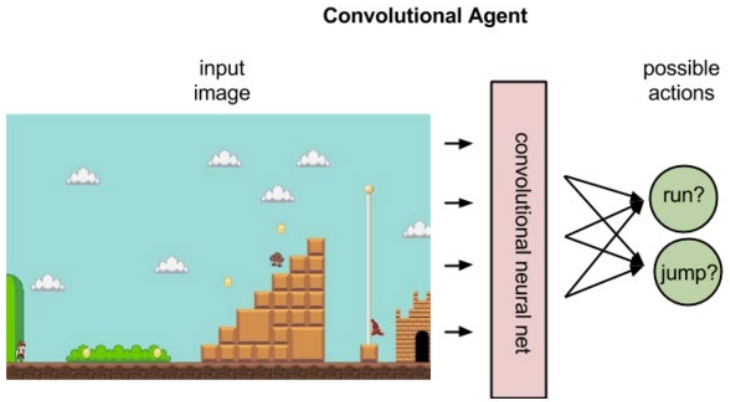
- Use deep neural networks to represent
  - Value function – Deep Q-Networks (DQN)
  - Policy
  - Model

- Addresses the issue of state-space explosion/curse of dimensionality in tabular q-learning

# DNNs

› convolutional networks can be used to recognize an agent's state, and learn to map it to best actions




Credit: skymind.ai

# Resources

- https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/ - An article about the design and training process of a DeepMind agent that beats expert StarCraft players
- https://huggingface.co/deep-rl-course/unit0/introduction - Great course implementing RL in various Atari environments

# RL examples

# Benchmarks - Gymnasium

- Cart pole
- Mountain car
  - Discrete, continuous, multi-objective
- Taxi
- gridworld
  - Windy, frozen lake, lava world