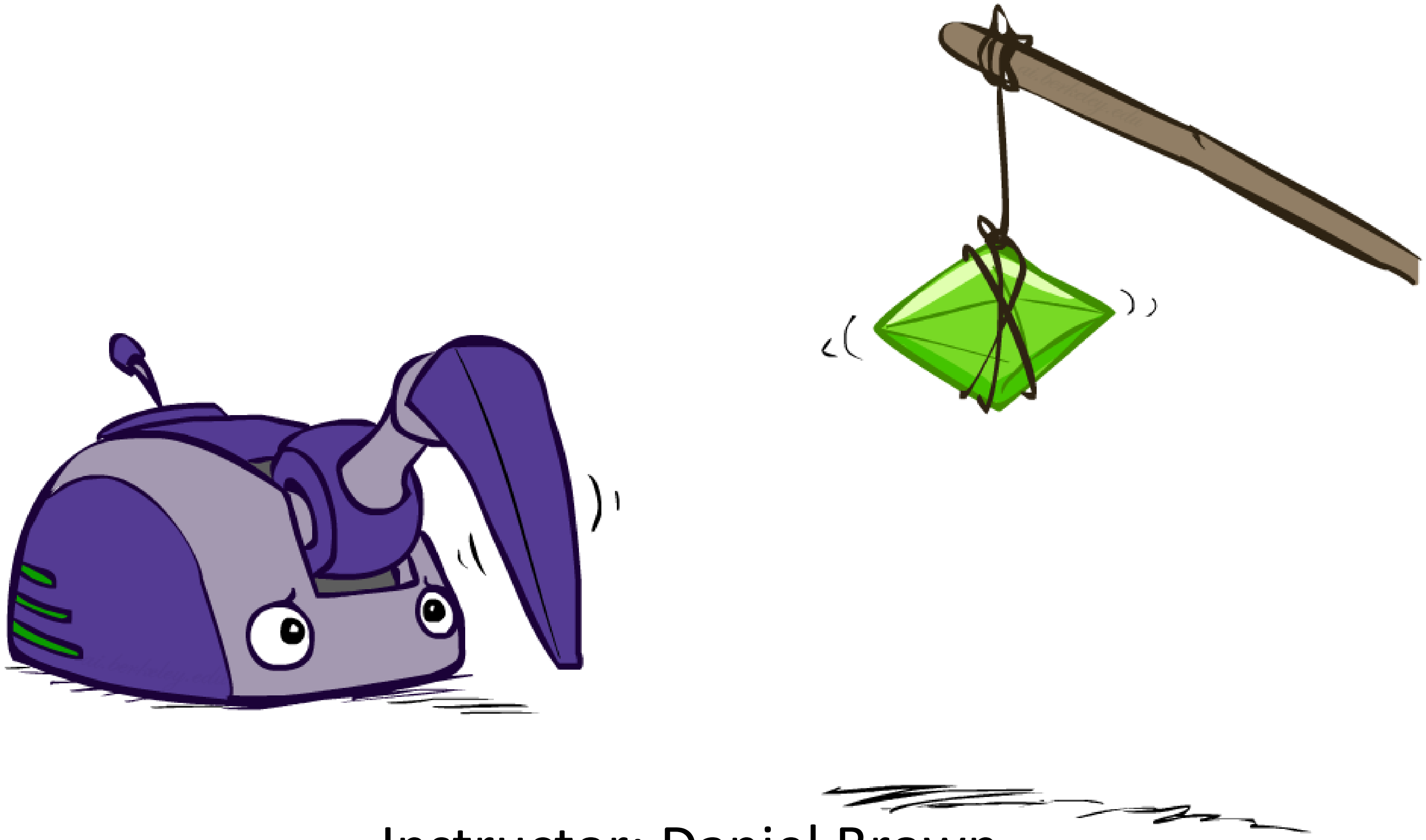
Intro to Value-Based Reinforcement Learning



Instructor: Daniel Brown
University of Utah

[Some content borrowed from slides created by Dan Klein and Pieter Abbeel http://ai.berkeley.edu.]

What changes?

- Rather than planning, we now need to learn!
  - No access to underlying MDP, can't solve it with just computation
  - You needed to actually act to figure it out
  - Extension and generalization of Multi-Armed Bandits
- 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

Example: Learning to Walk

Initial

A Learning Trial

After Learning [1K Trials]

[Kohl and Stone, ICRA 2004]

Example: Learning to Walk

Initial

[Kohl and Stone, ICRA 2004]

[Video: AIBO WALK – initial]

Example: Learning to Walk

Training

[Kohl and Stone, ICRA 2004]

[Video: AIBO WALK – training]

Example: Learning to Walk

Finished

[Kohl and Stone, ICRA 2004]

[Video: AIBO WALK – finished]

https://vision-locomotion.github.io/

Startups

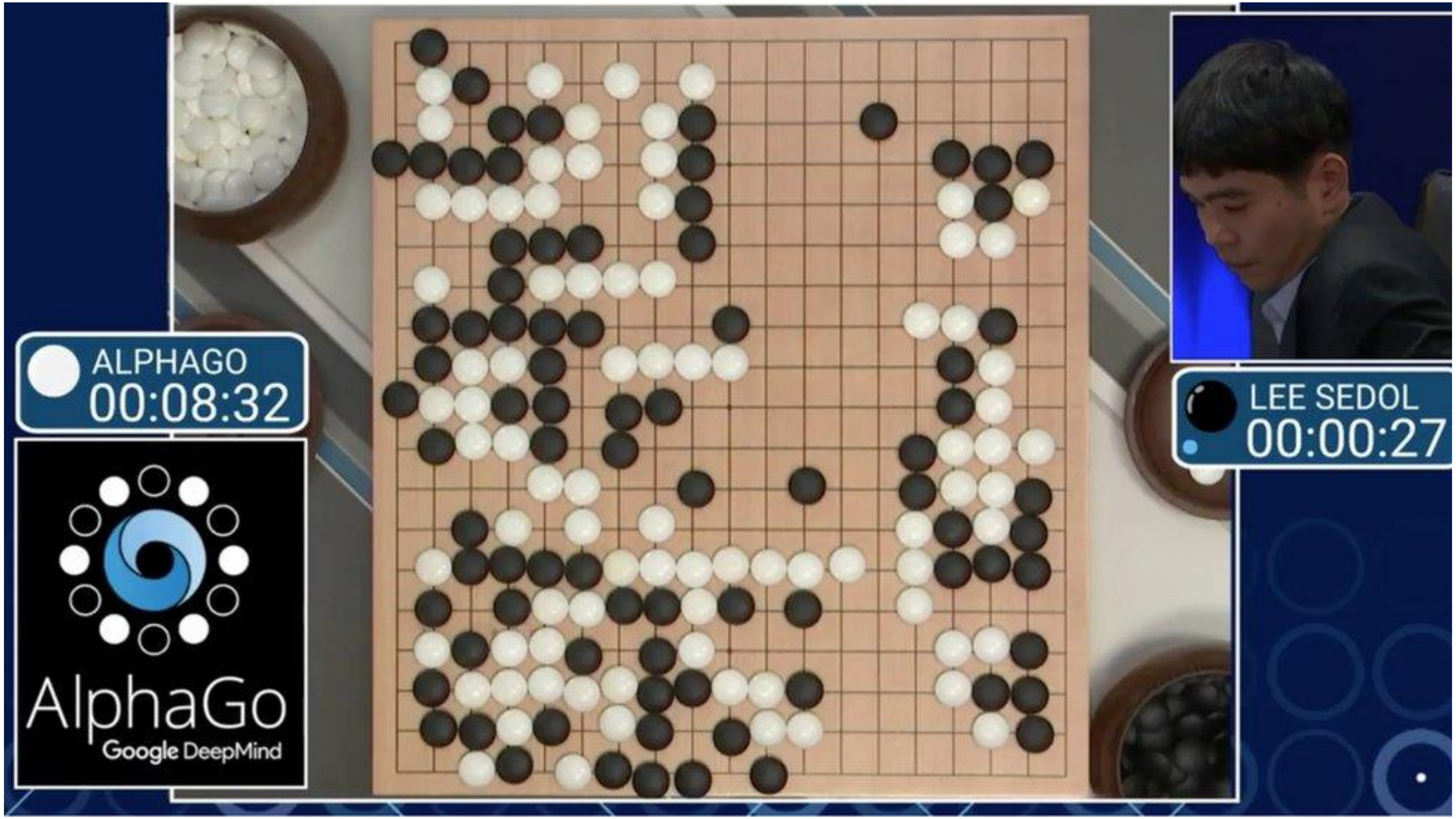
Google Acquires Artificial Intelligence Startup DeepMind For More Than $500M

Catherine Shu @catherineshu / 6:20 PM MST • January 26, 2014

Comment

The Arcade Learning Environment

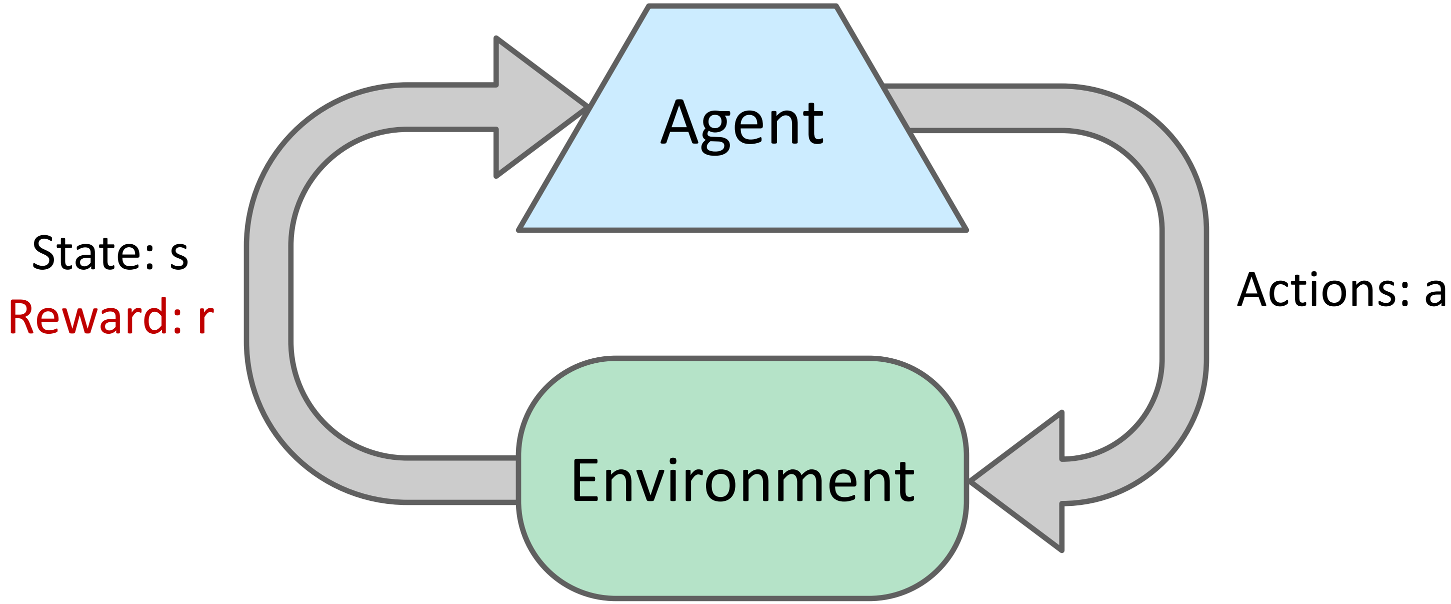

021
3
1


ALPHAGO
00:08:32
AlphaGo
Google DeepMind
LEE SEDOL
00:00:27

ChatGPT

deepseek

Reinforcement Learning



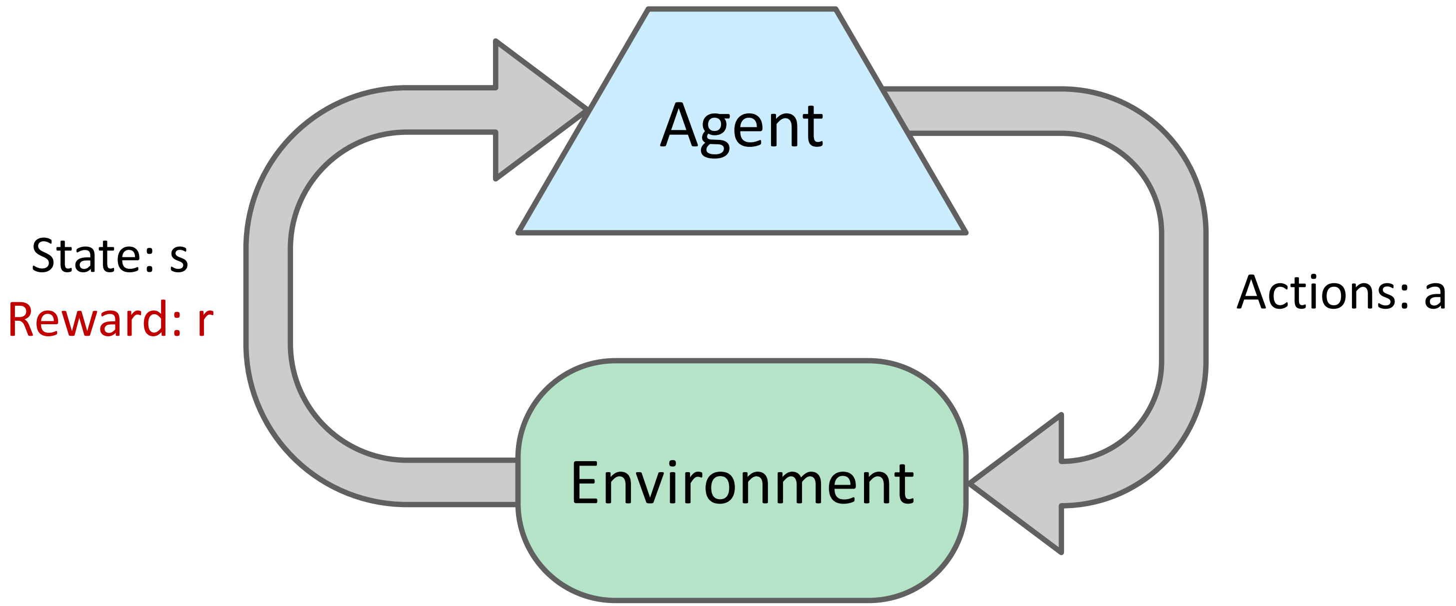
- Basic idea:
  - Receive feedback in the form of rewards
  - Agent's utility is defined by the reward function
  - Must (learn to) act so as to maximize expected rewards
  - All learning is based on observed samples of outcomes!

Why Reinforcement Learning?

- Takes inspiration from nature
- Often easier to encode a task as a sparse reward (e.g. recognize if goal is achieved) but hard to hand-code how to act so reward is maximized (e.g. Go)
- General purpose AI framework

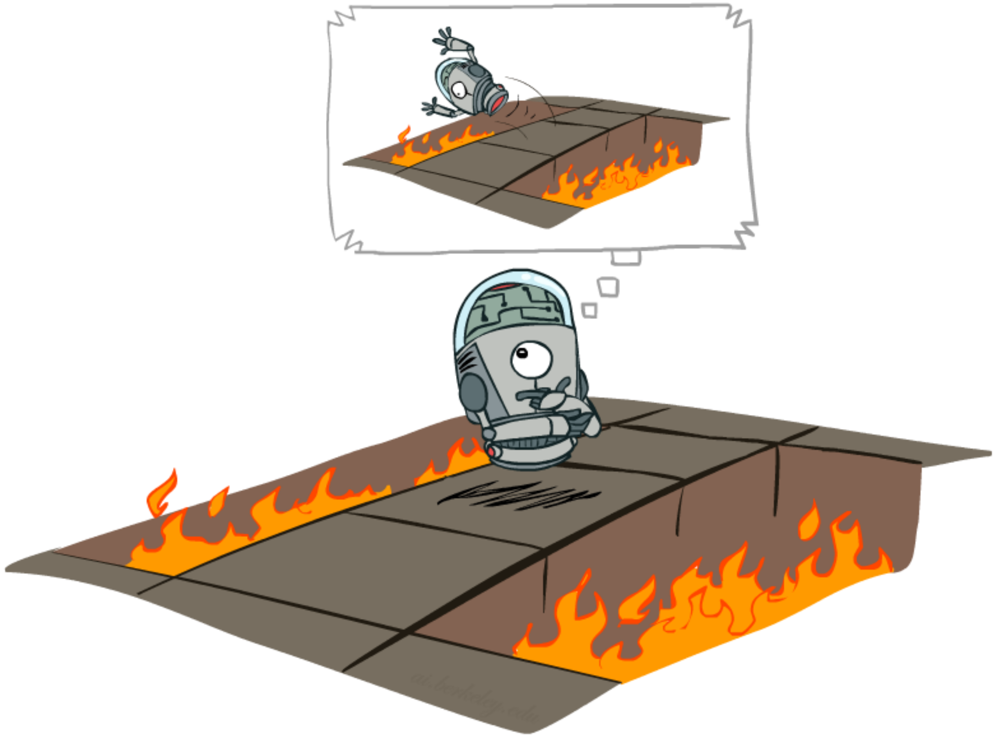
Reinforcement Learning

- 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
  - I.e. we don't know which states are good or what the actions do
  - Must actually try actions and states out to learn

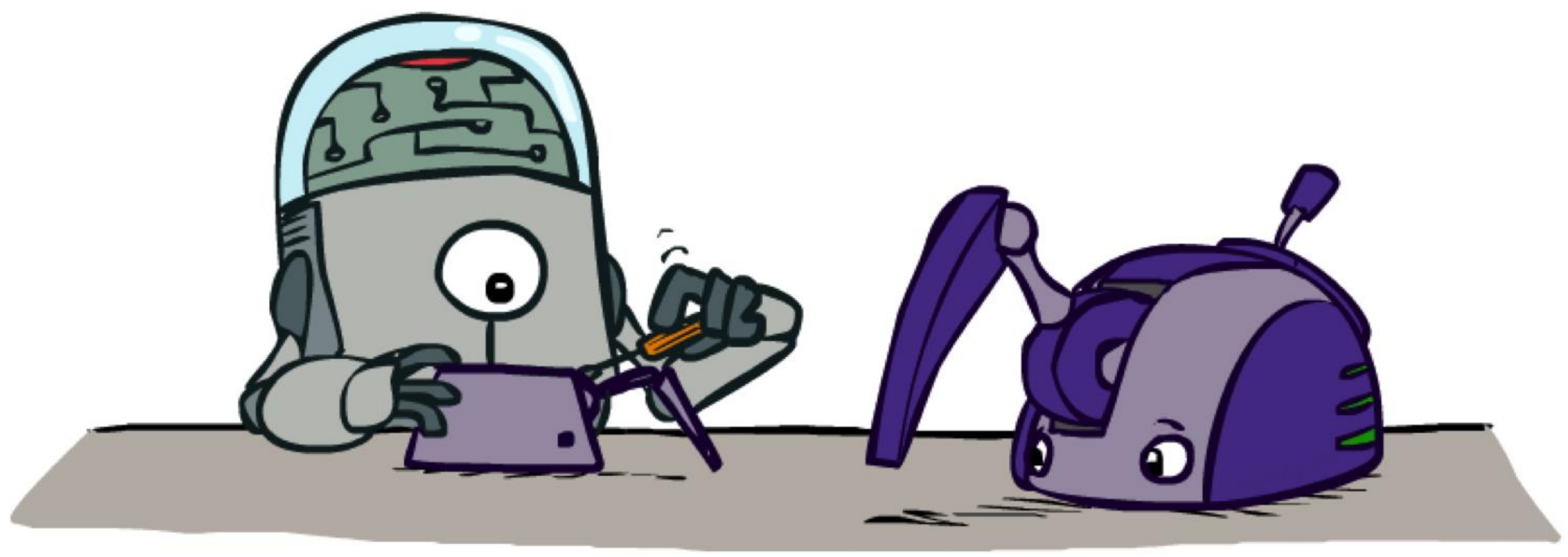
Offline (MDPs) vs. Online (RL)



Offline Solution

Online Learning

Model-Based Learning



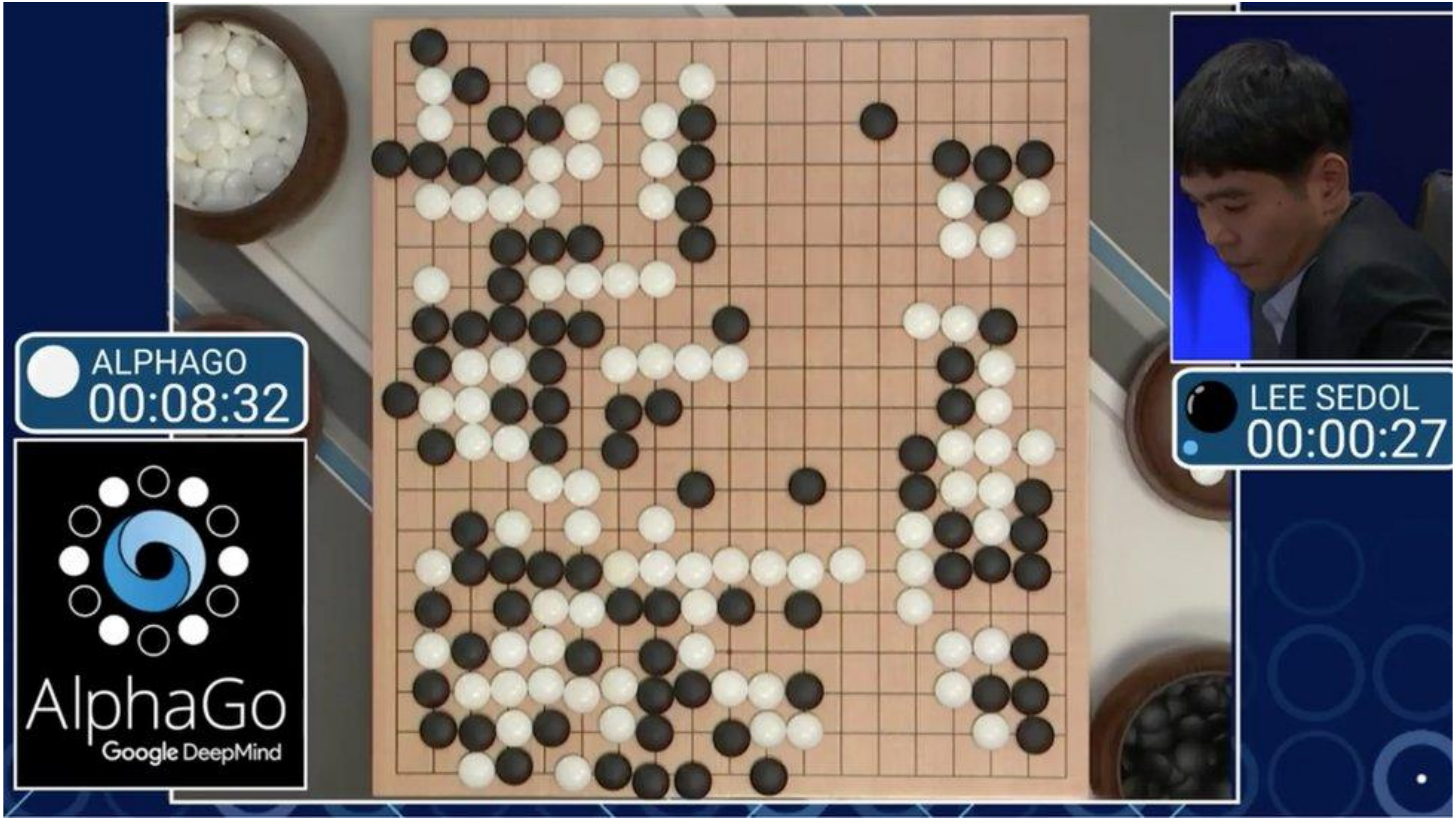
Simple View of Model-Based RL

- 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 $\widehat{T}(s, a, s')$
  - Discover each $\widehat{R}(s, a, s')$ when we experience (s, a, s')

- Step 2: Solve the learned MDP
  - For example, use value iteration, as before

Sometimes Model of World is Known



Deep RL Makes a Big Splash!

nature

Published: 25 February 2015

Human-level control through deep reinforcement learning

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg & Demis Hassabis

When might RL be a good tool for your problem?

When might RL be a good tool for your problem?

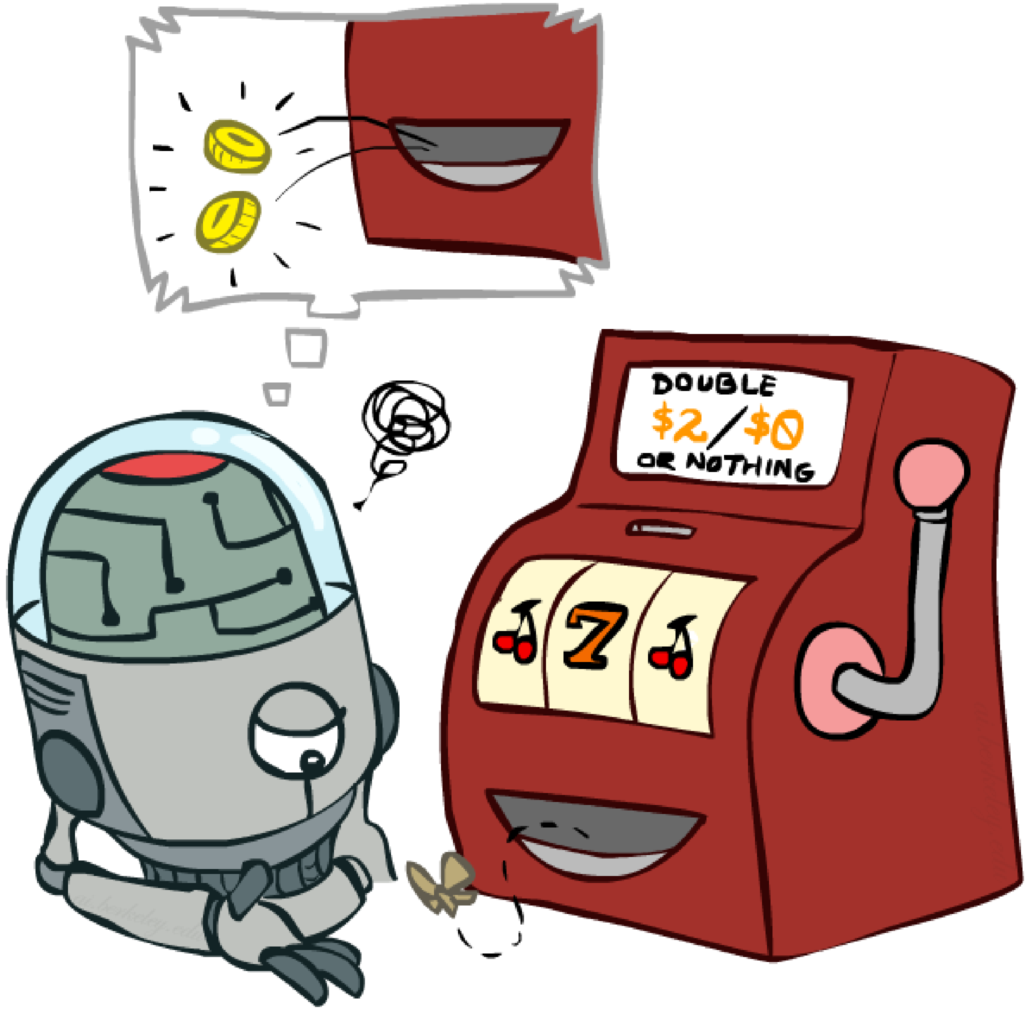
- Is your problem a sequential decision making problem?
- Are there “actions” that effect the next “state”?
- Do you know the rules of these effects?
- Can you write down a clear objective/score/reward/cost?
- Do you have a simulator?
- Lots of examples of sequences of decisions and their long-term consequences?
- Is it unclear what to do in each state? Exploration required?
- Are you looking for unique/creative/super-human solutions?

When might RL not be a good tool?

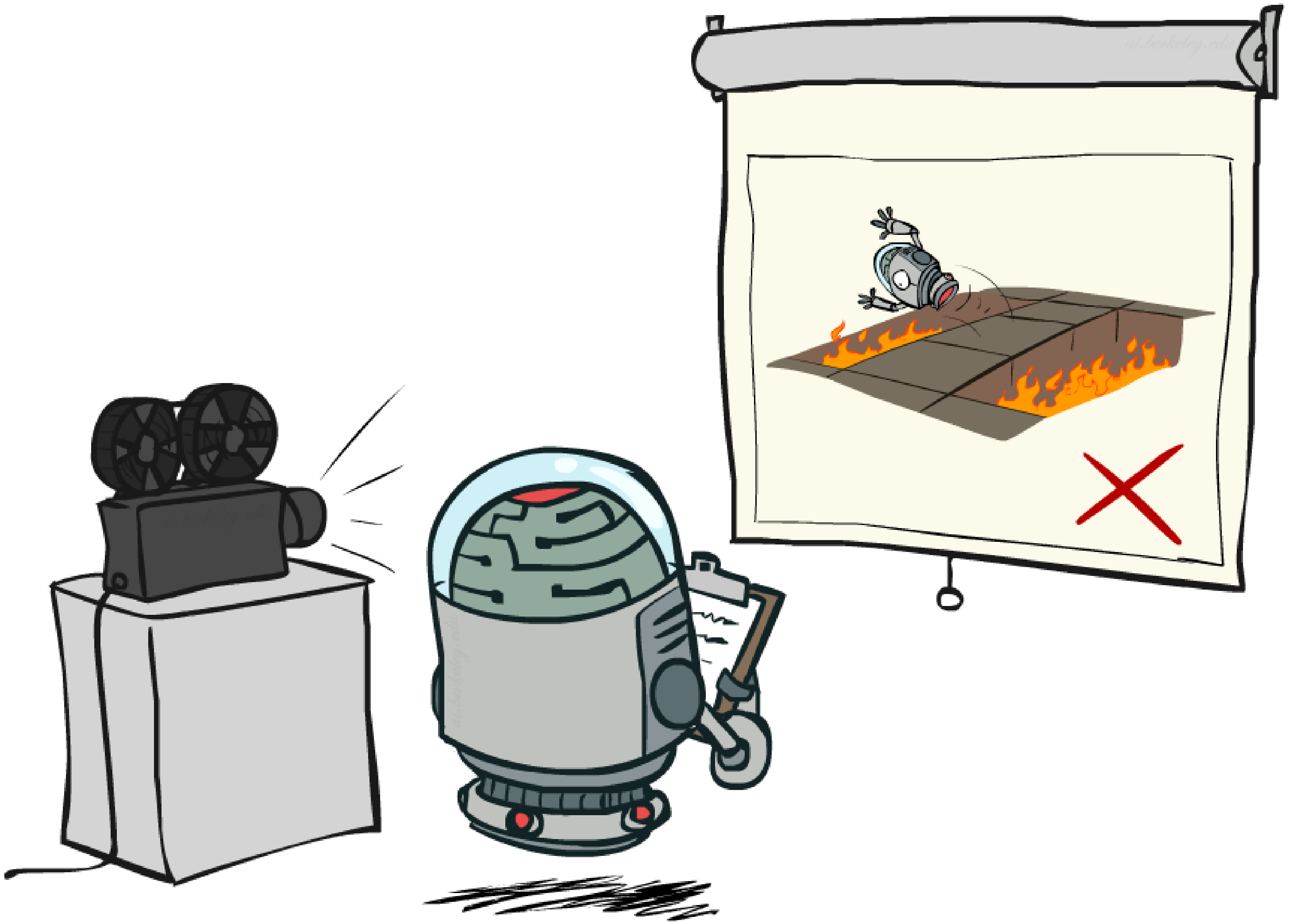
When might RL not be a good tool?

- Single step or static problem
- No clear reward signal.
- Reward signal is unavailable or very hard to write down.
- Well-known model of the environment.
- Deterministic environment
- Low-tolerance for exploration and trial and error
- No need for adaptive or novel solutions. The goal is to perform the task in a very predictable way.

Model-Free Learning

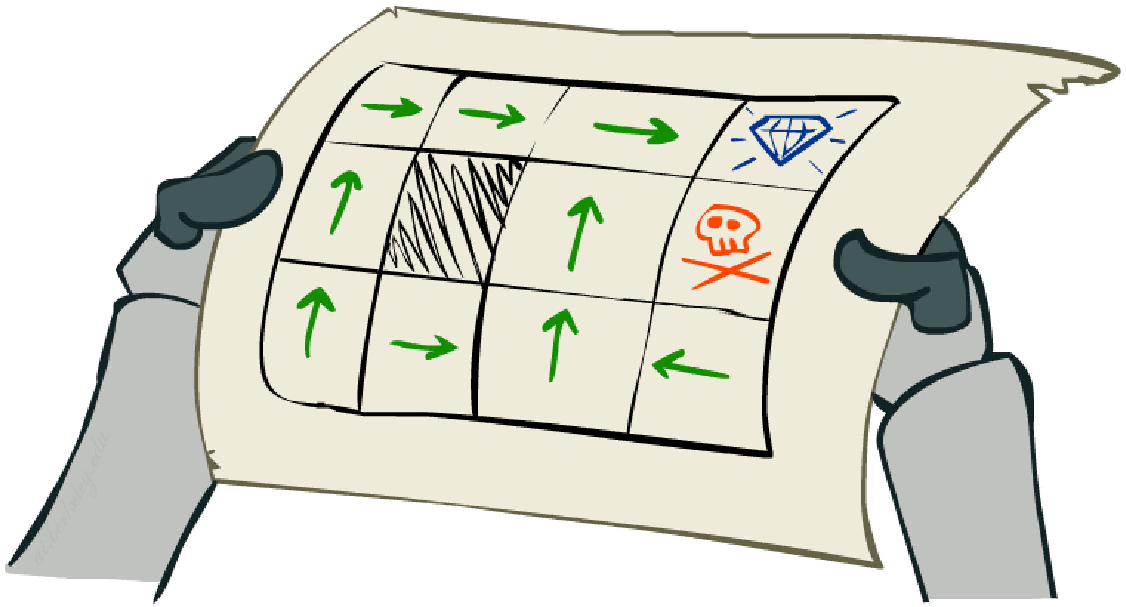


Passive Reinforcement Learning



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 (Monte Carlo 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

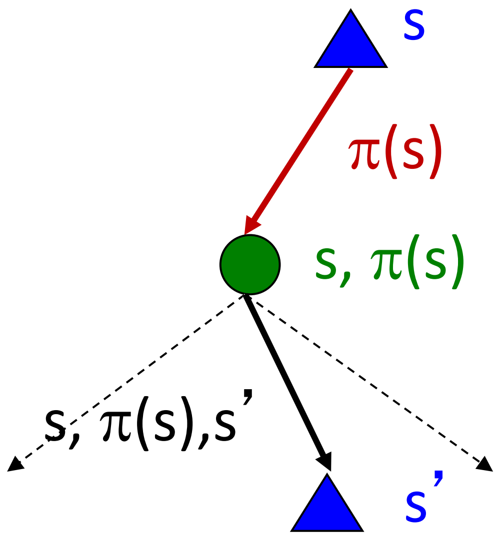
Problems with Direct Evaluation

- 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

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?

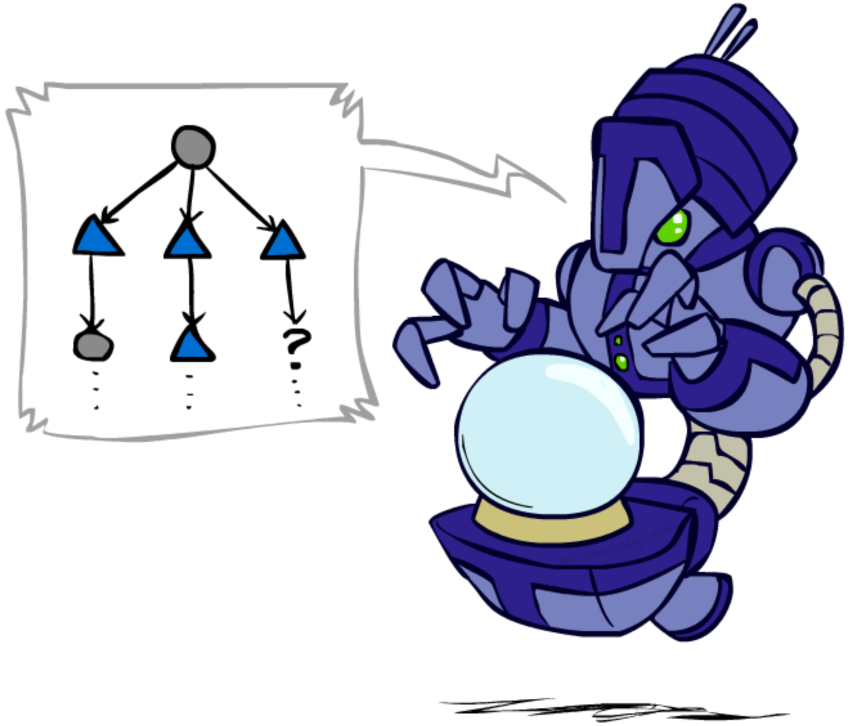
Sample-Based Policy Evaluation?

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

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

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

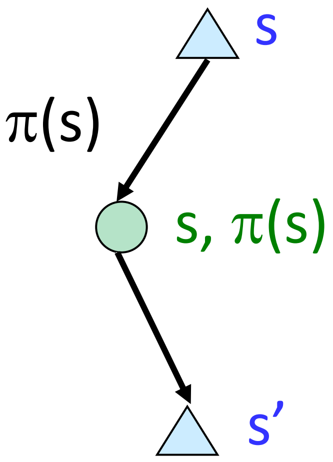
$$sample_1 = R(s, \pi(s), s_1') + \gamma V_k^{\pi}(s_1')$$
$$sample_2 = R(s, \pi(s), s_2') + \gamma V_k^{\pi}(s_2')$$
$$\ldots$$
$$sample_n = R(s, \pi(s), s_n') + \gamma V_k^{\pi}(s_n')$$



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

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:

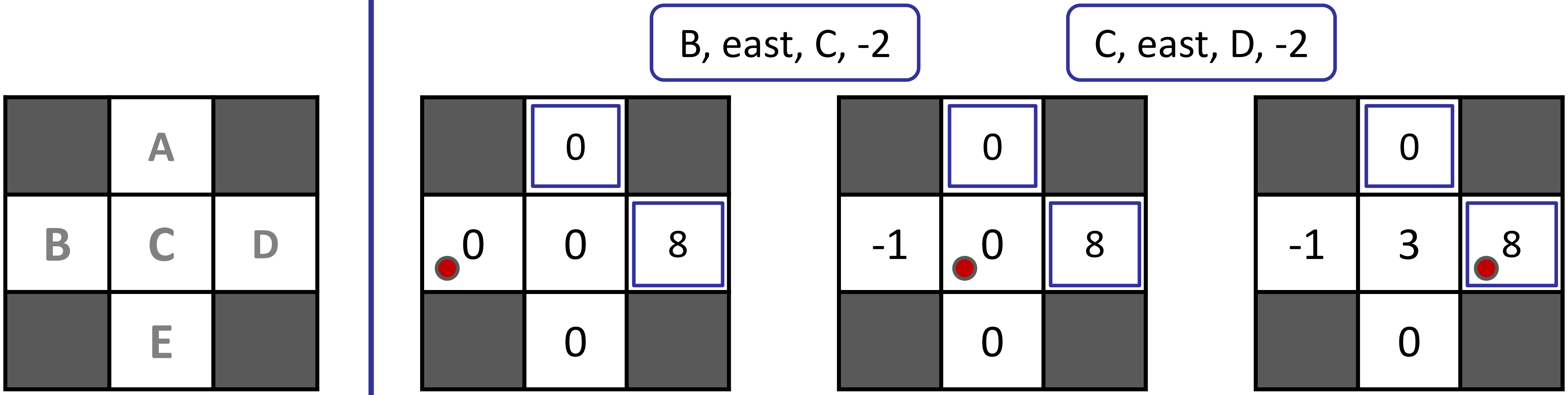
$$\bar{x}_n = \frac{x_n + (1-\alpha) \cdot x_{n-1} + (1-\alpha)^2 \cdot x_{n-2} + \dots}{1 + (1-\alpha) + (1-\alpha)^2 + \dots}$$

  - Forgets about the past (distant past values were wrong anyway)
- Decreasing learning rate (alpha) can give converging averages

Example: Temporal Difference Learning

States

Observed Transitions



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

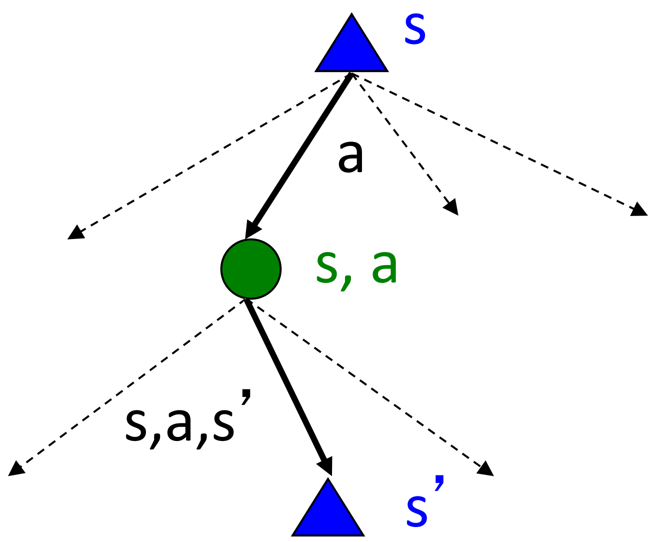
$$V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + \alpha\left[R(s,\pi(s),s') + \gamma V^{\pi}(s')\right]$$

Problems with TD Value 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:

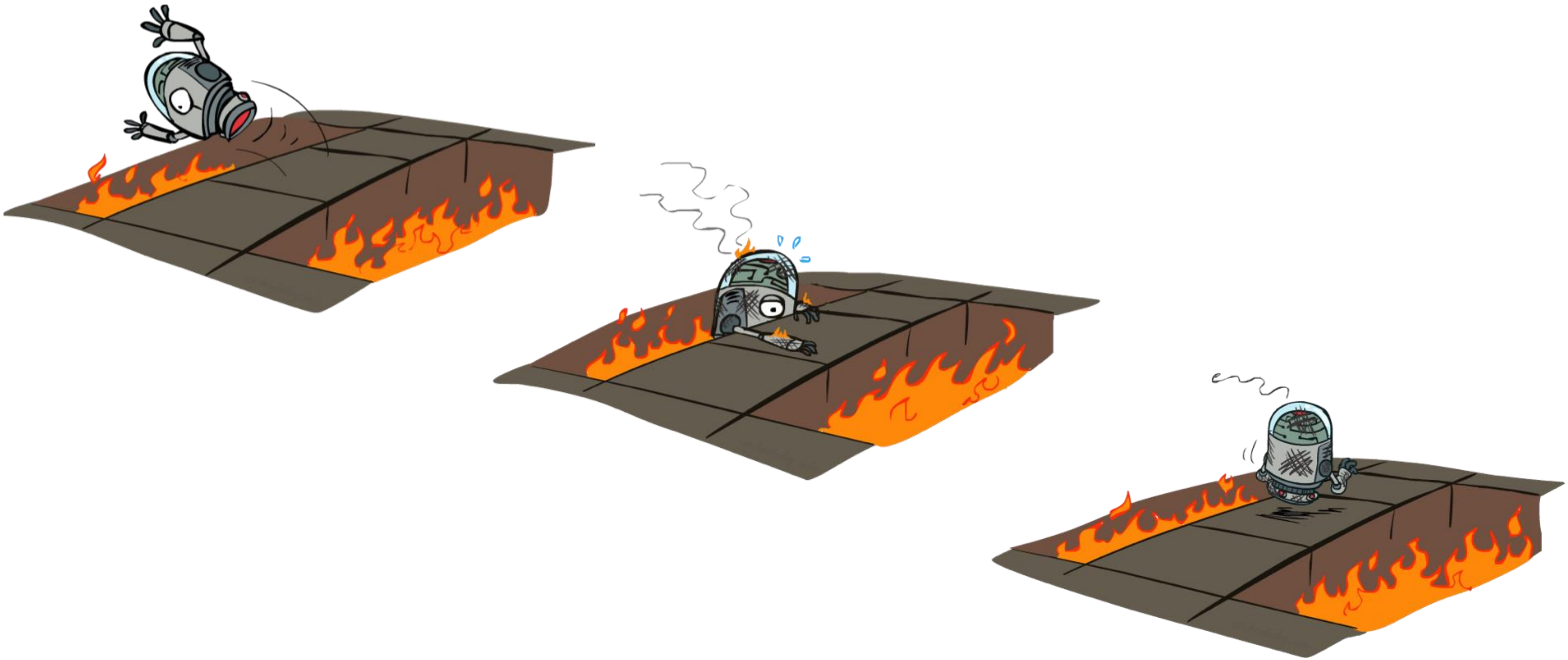
$$\pi(s) = \arg\max_a Q(s,a)$$

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



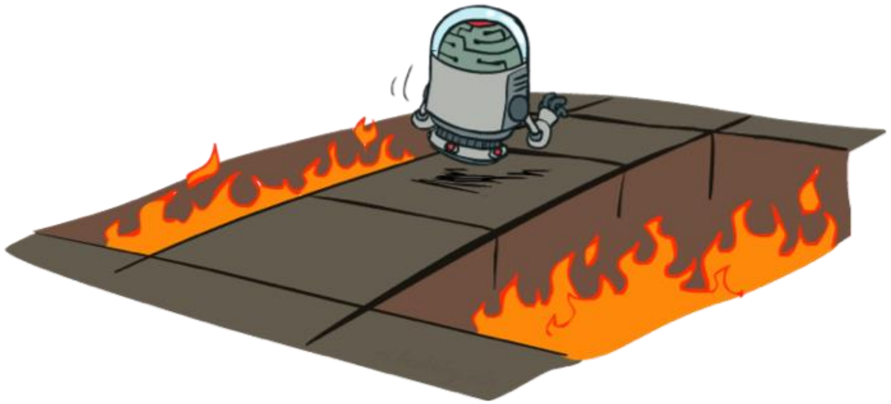
- Idea: learn Q-values, not values
- Makes action selection model-free too!

Active Reinforcement Learning



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…

Detour: 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:

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

- Can we write out a bellman equation like value iteration, but only using Q values?

Detour: 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

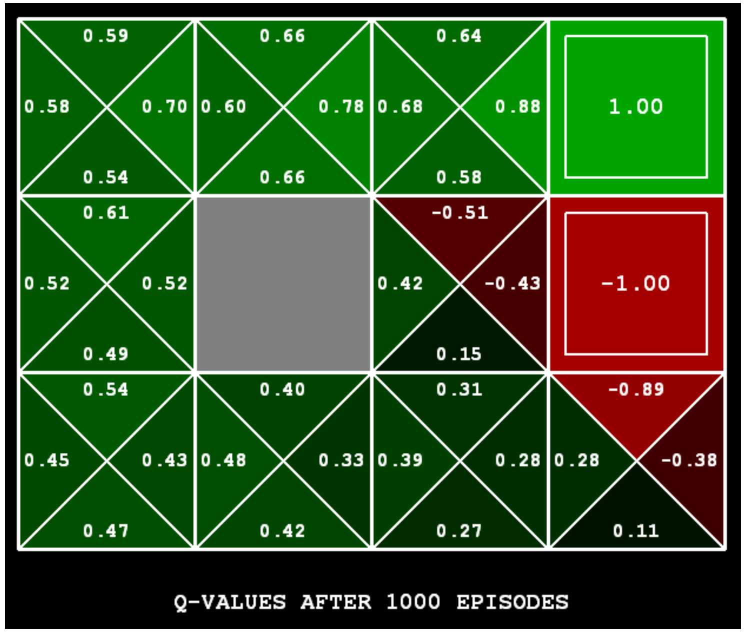
$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a') \right]$$

- 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:

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



[Demo: Q-learning – gridworld (L10D2)]
[Demo: Q-learning – crawler (L10D3)]

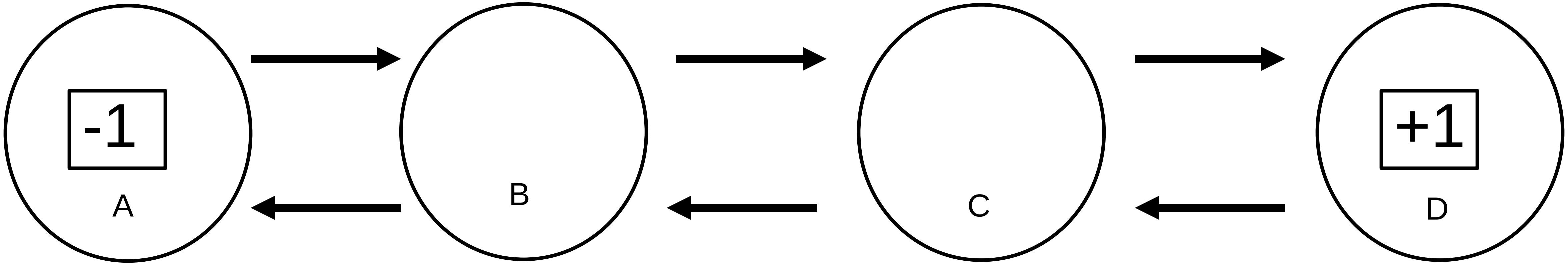
Example

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

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

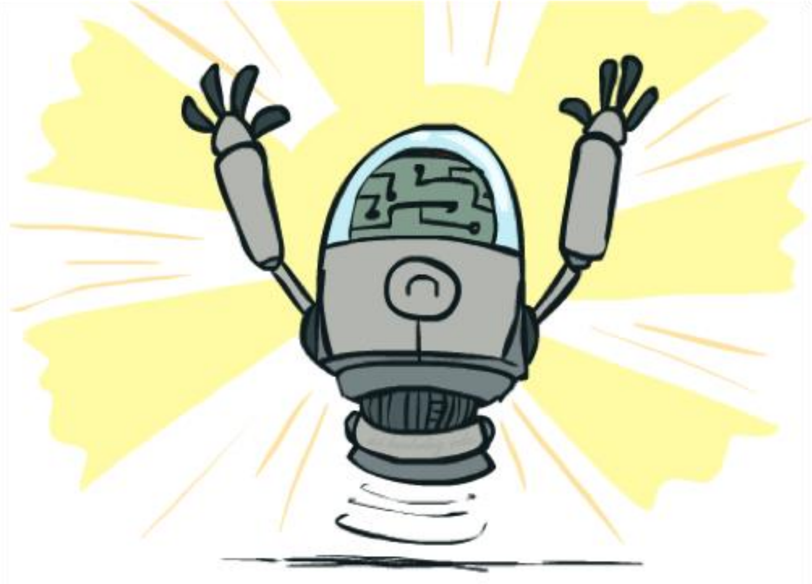
$\alpha = \frac{1}{2}, \gamma = 1.$

Experience: (D,exit, terminal, +1), (C,->,D,0)



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 (!)

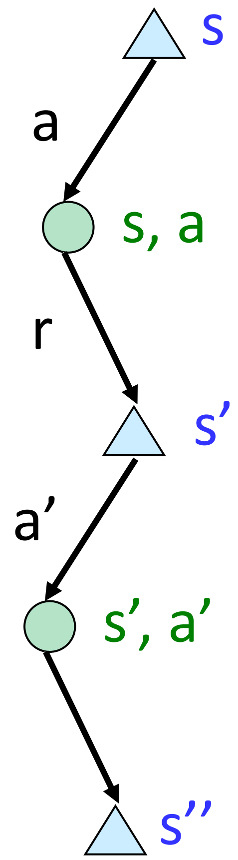


Model-Free Learning

- Model-free (temporal difference) learning
  - Experience world through episodes

$$(s, a, r, s', a', r', s'', a'', r'', s'''' \ldots)$$

  - Update estimates each transition (s, a, r, s')
  - Over time, updates will mimic Bellman updates



Q-Learning Recap

- We'd like to do Q-value updates to each Q-state:

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

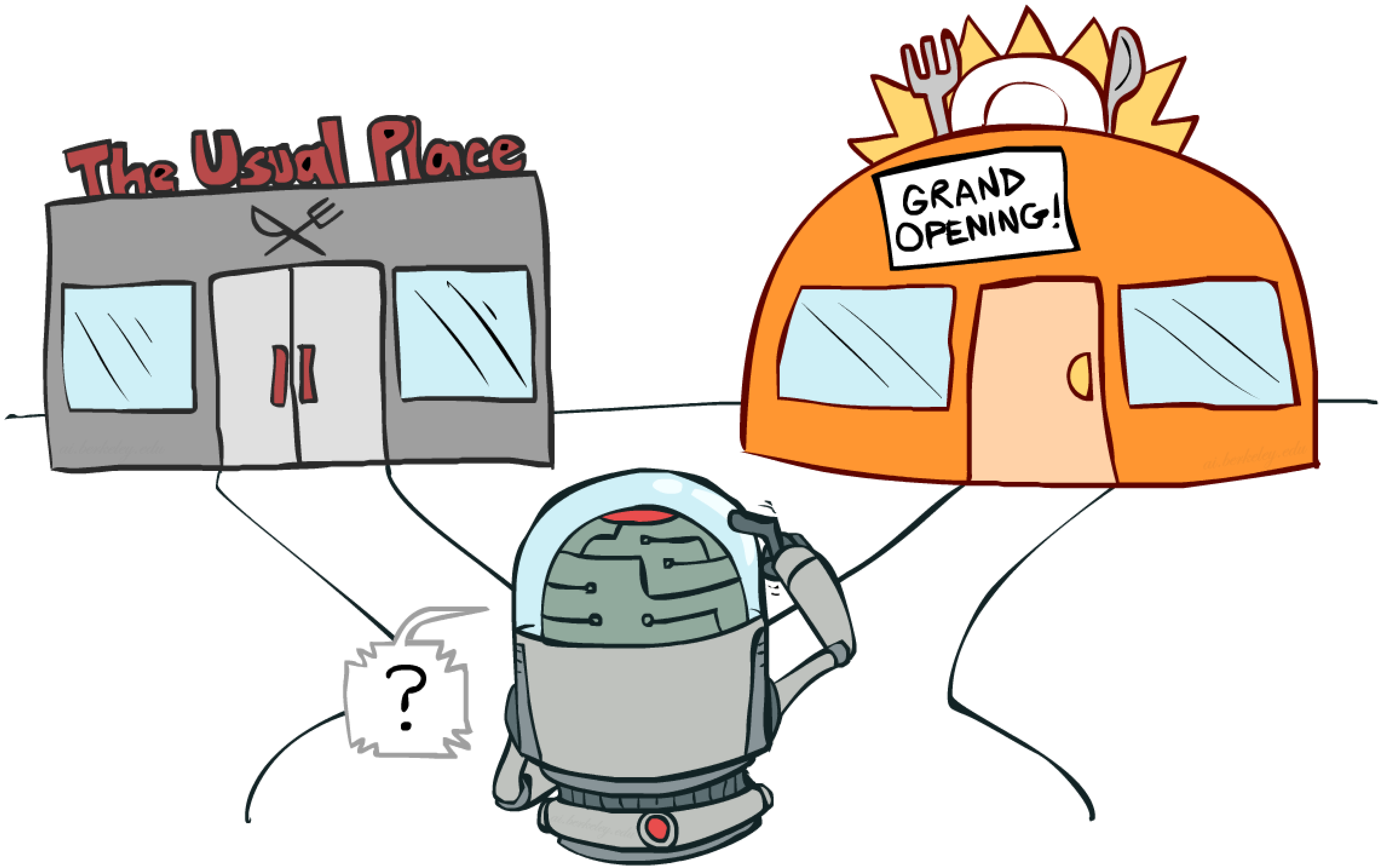
  - But can't compute this update without knowing T, R
- Instead, compute average as we go
  - Receive a sample transition (s,a,r,s')
  - This sample suggests $Q(s,a) \approx r + \gamma \max_{a'} Q(s',a')$
  - But we want to average over results from (s,a) (Why?)
  - So keep a running average

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

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') \ - Q(s,a))$$

Useful alternate form of update for Q-learning. We want to push the Q-value towards the sample!

Exploration vs. Exploitation



How to Explore?

- 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

[Demo: Q-learning – manual exploration – bridge grid (L11D2)]
[Demo: Q-learning – epsilon-greedy -- crawler (L11D3)]

Exploration Functions

- When to explore?
  - Random actions: explore a fixed amount
  - Better idea: explore areas whose badness is not (yet) established, eventually stop exploring

- Exploration function
  - Takes a value estimate u and a visit count n, and returns an optimistic utility, e.g. $f(u,n) = u + k/n$

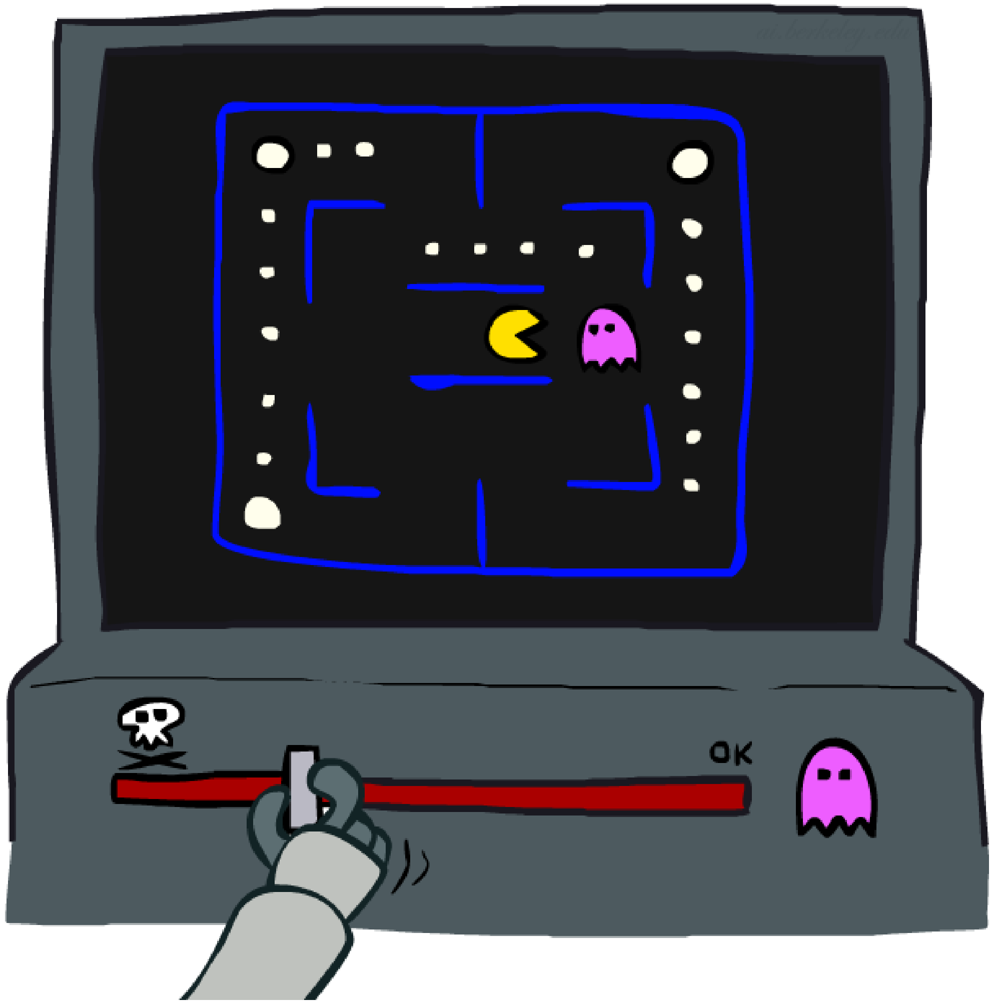
Regular Q-Update: $Q(s,a) \leftarrow_\alpha R(s,a,s') + \gamma \max_{a'} Q(s',a')$

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

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

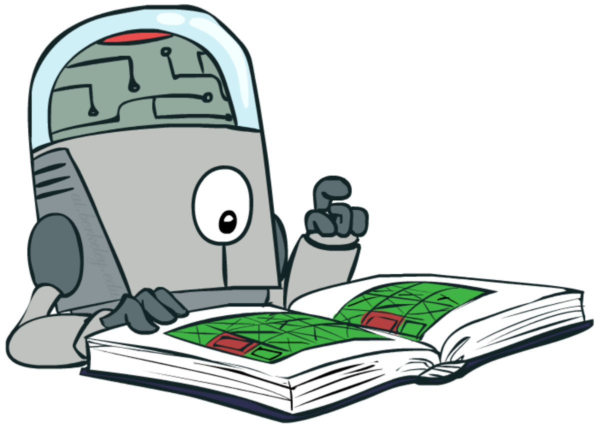
[Demo: exploration – Q-learning – crawler – exploration function (L11D4)]

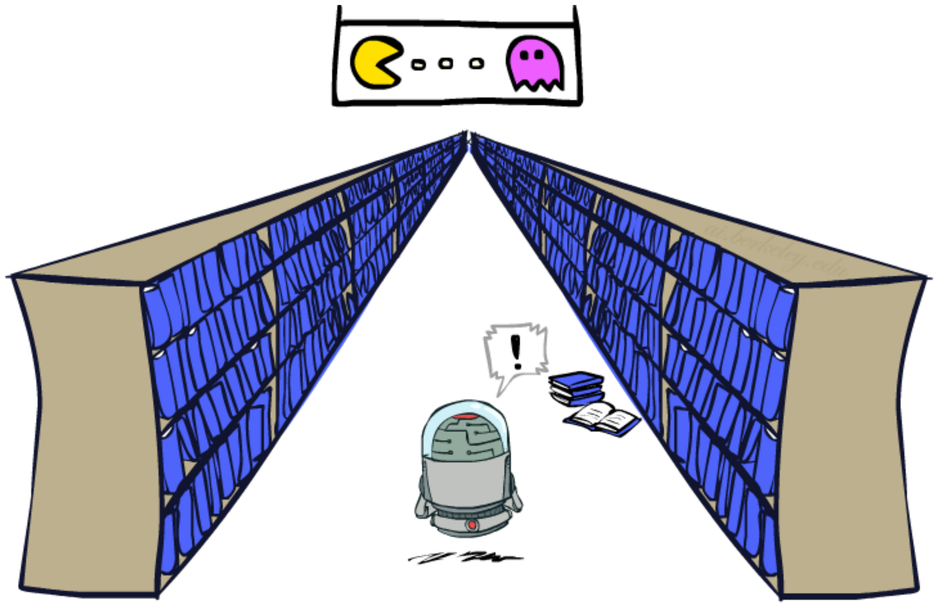
Approximate Q-Learning



Generalizing Across States

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

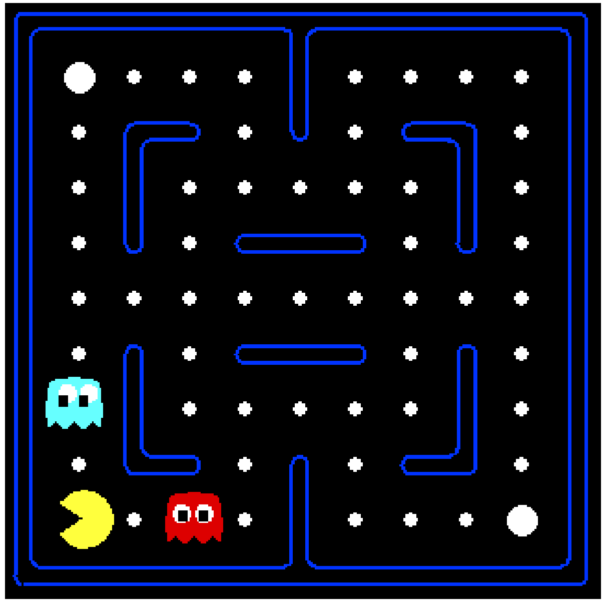




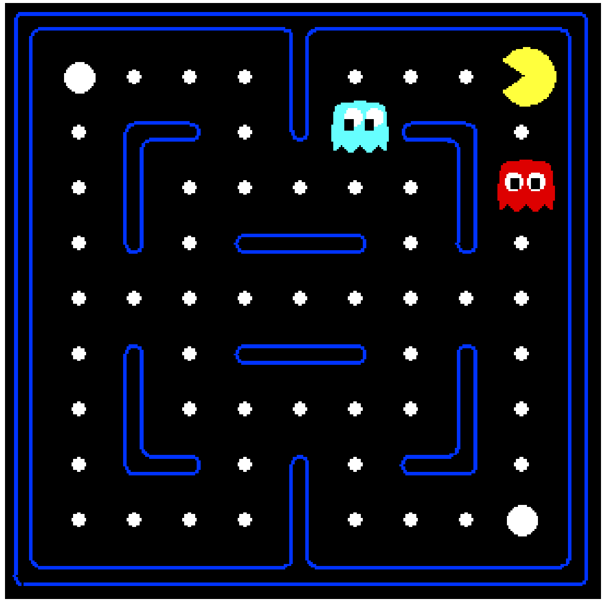
[demo – RL pacman]

Example: Pacman

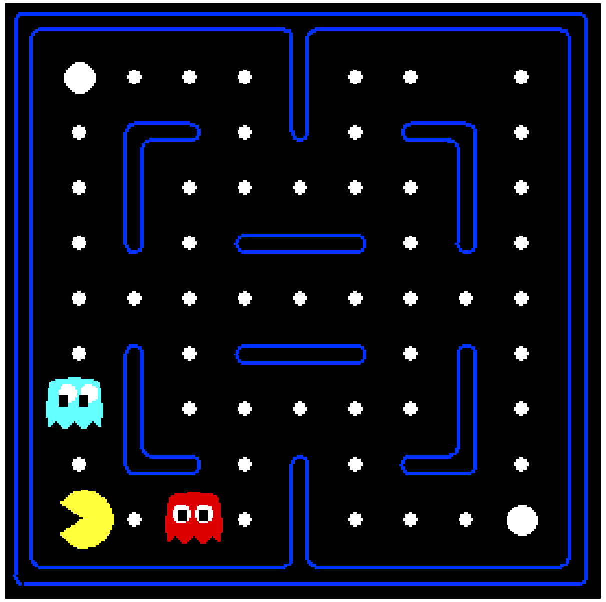
Let's say we discover through experience that this state is bad:



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

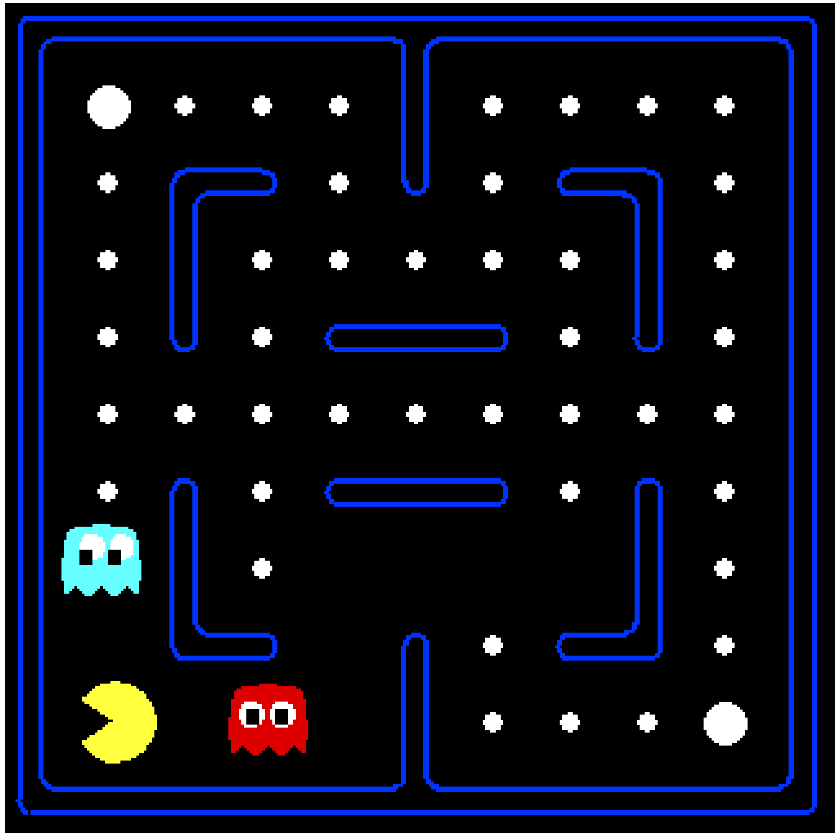


Or even this one!



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

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

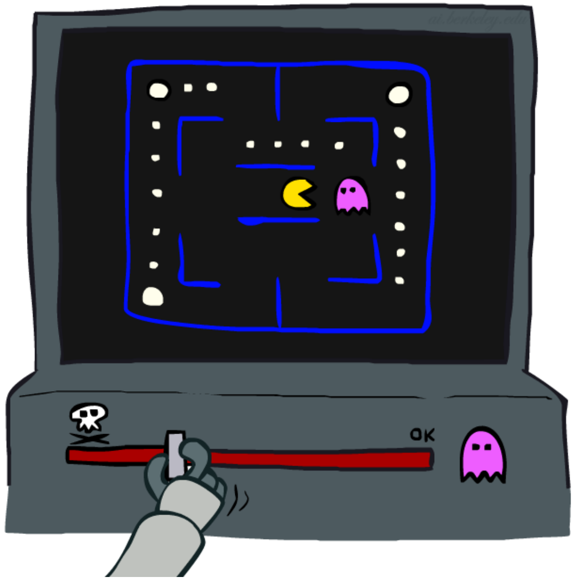
- Q-learning with linear Q-functions:

  transition $= (s, a, r, s')$

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

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

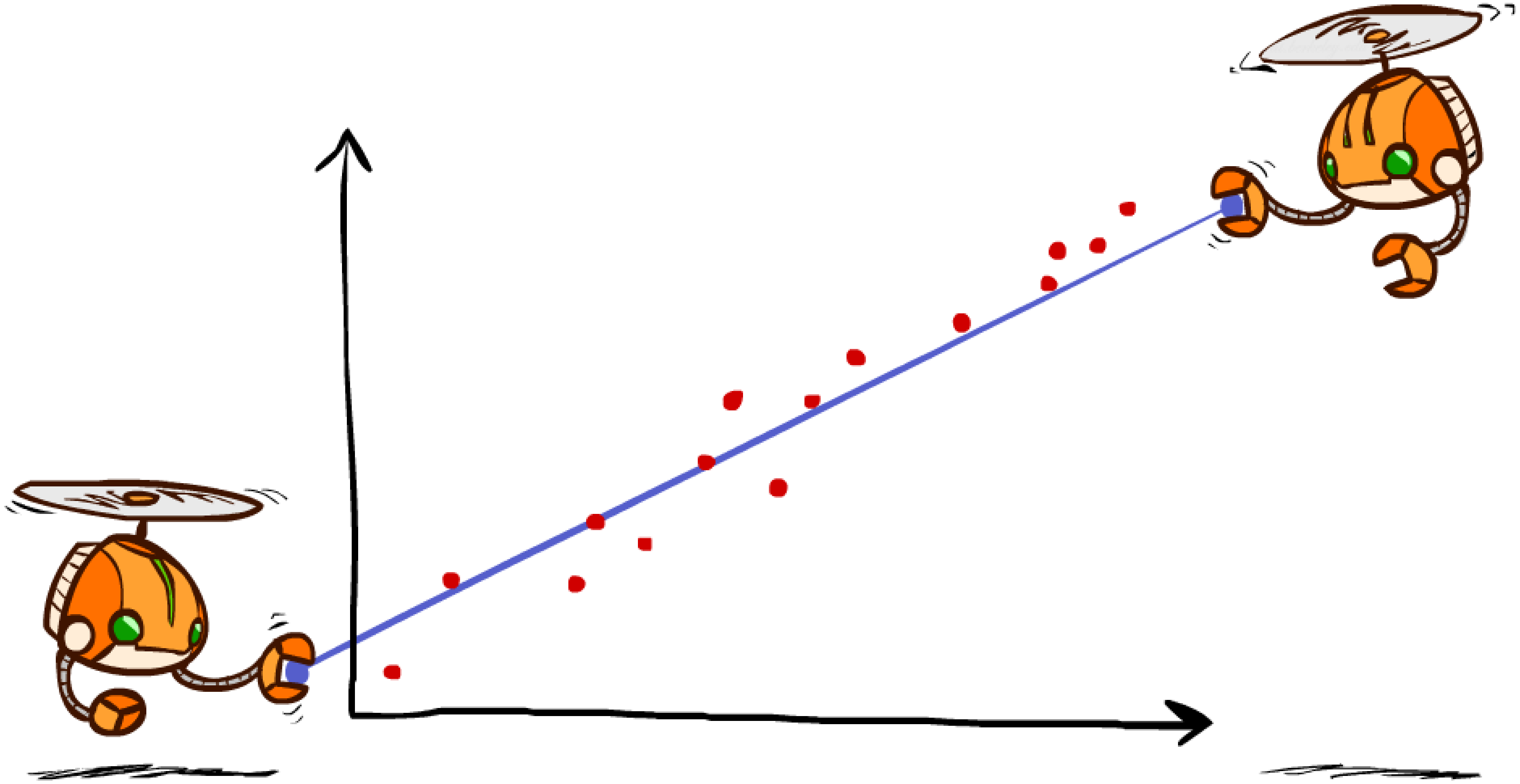
  $w_i \leftarrow w_i + \alpha$ [difference] $f_i(s,a)$ Approximate Q's



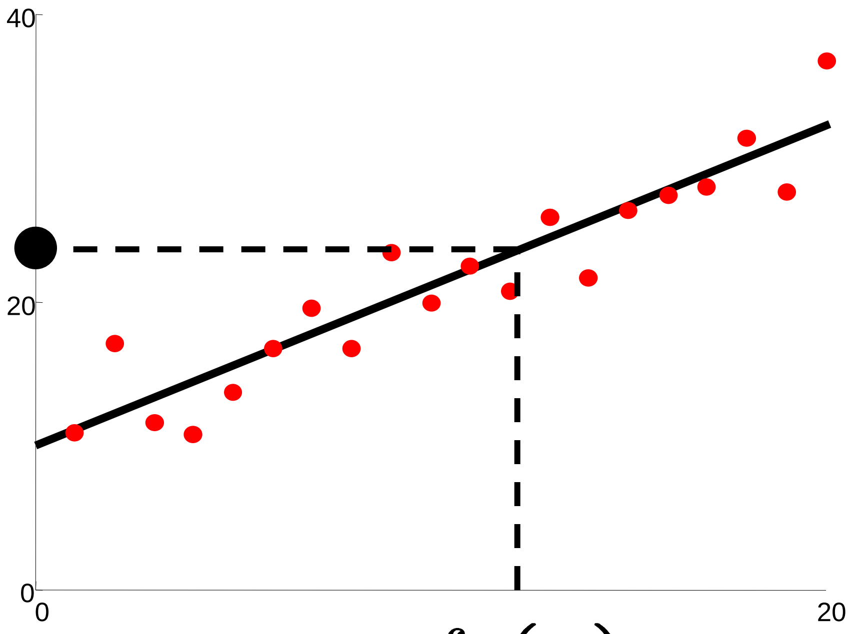
- Intuitive interpretation:
  - Adjust weights of active features
  - E.g., if something unexpectedly bad happens, blame the features that were on: disprefer all states with that state's features

- Formal justification: online least squares

Q-Learning and Least Squares

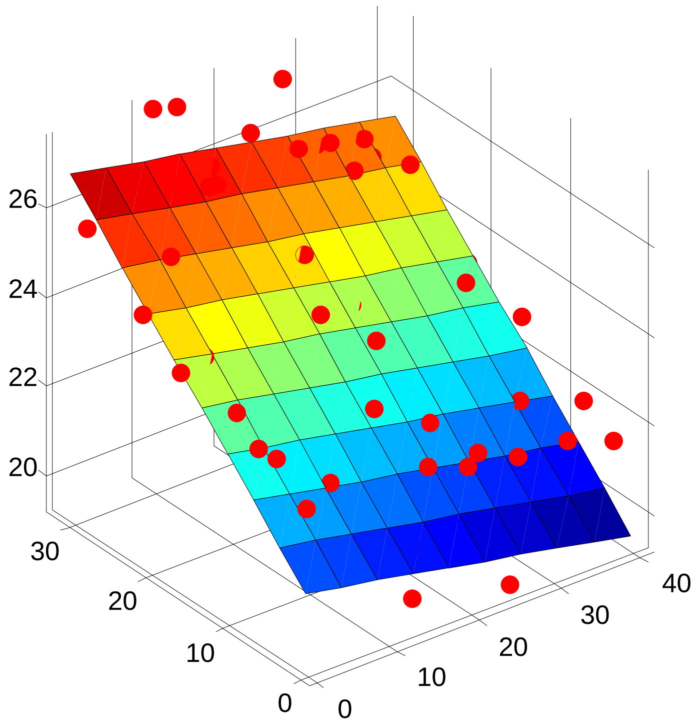


Linear Approximation: Regression



$$\widehat{y}$$

$$f_1(x)$$



Prediction:

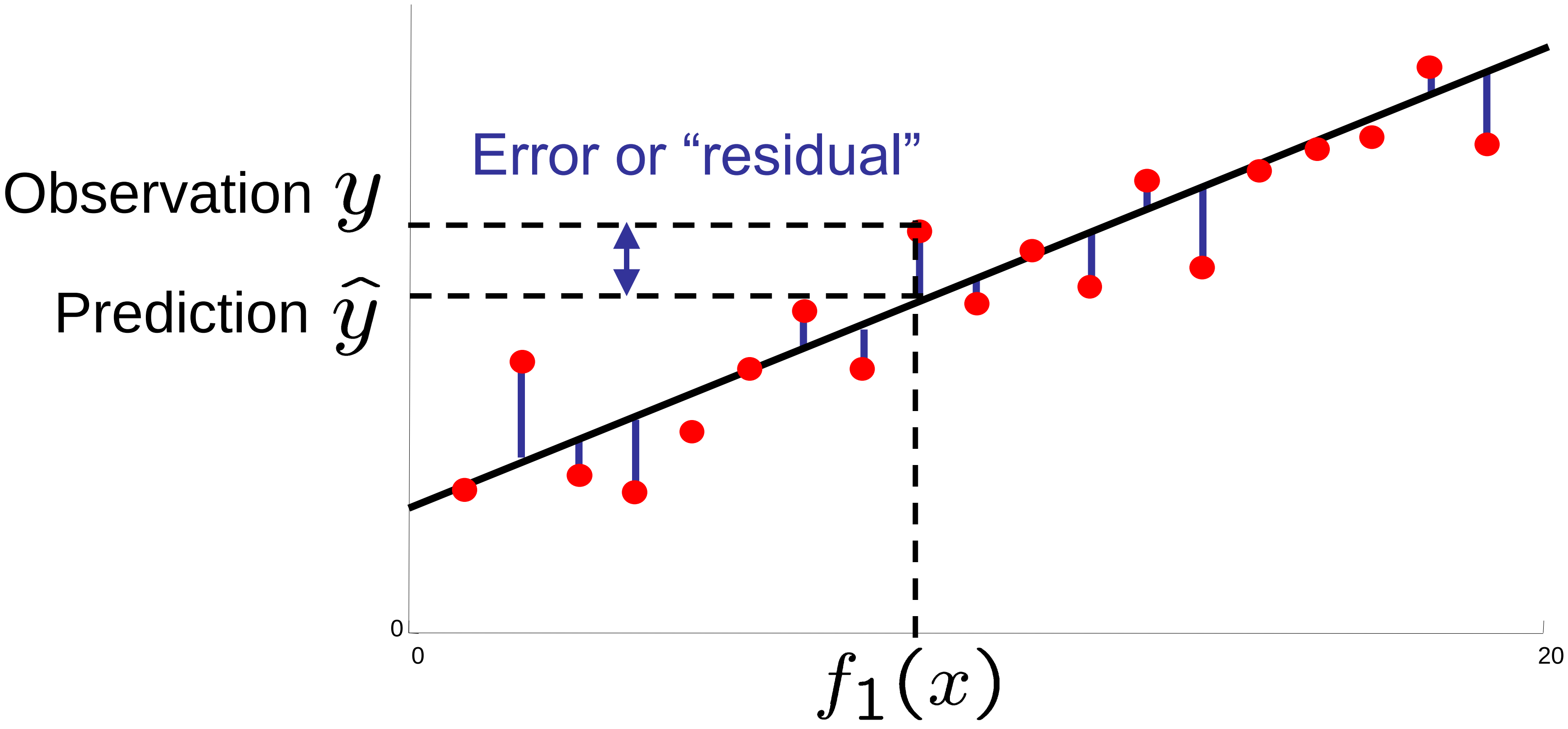
$$\widehat{y} = w_0 + w_1 f_1(x)$$

Prediction:

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

Optimization: Least Squares

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



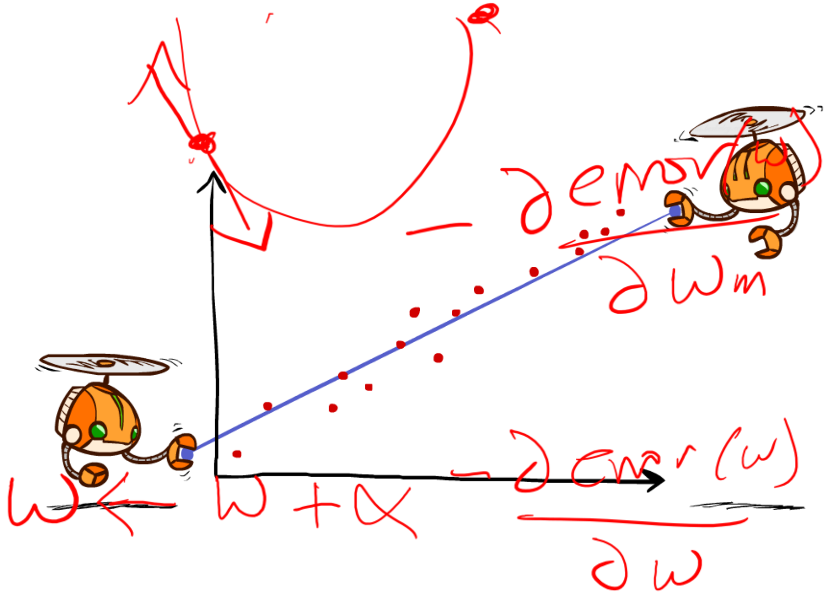
Minimizing Error

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”

Tabular Q-Learning is Special Case

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

"target" "prediction"

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

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

If feature is just an indicator for (s,a), then we recover the original tabular setting.

Non-linear function approximation

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

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

v.s.

$$V(s) = f_\theta(s)$$

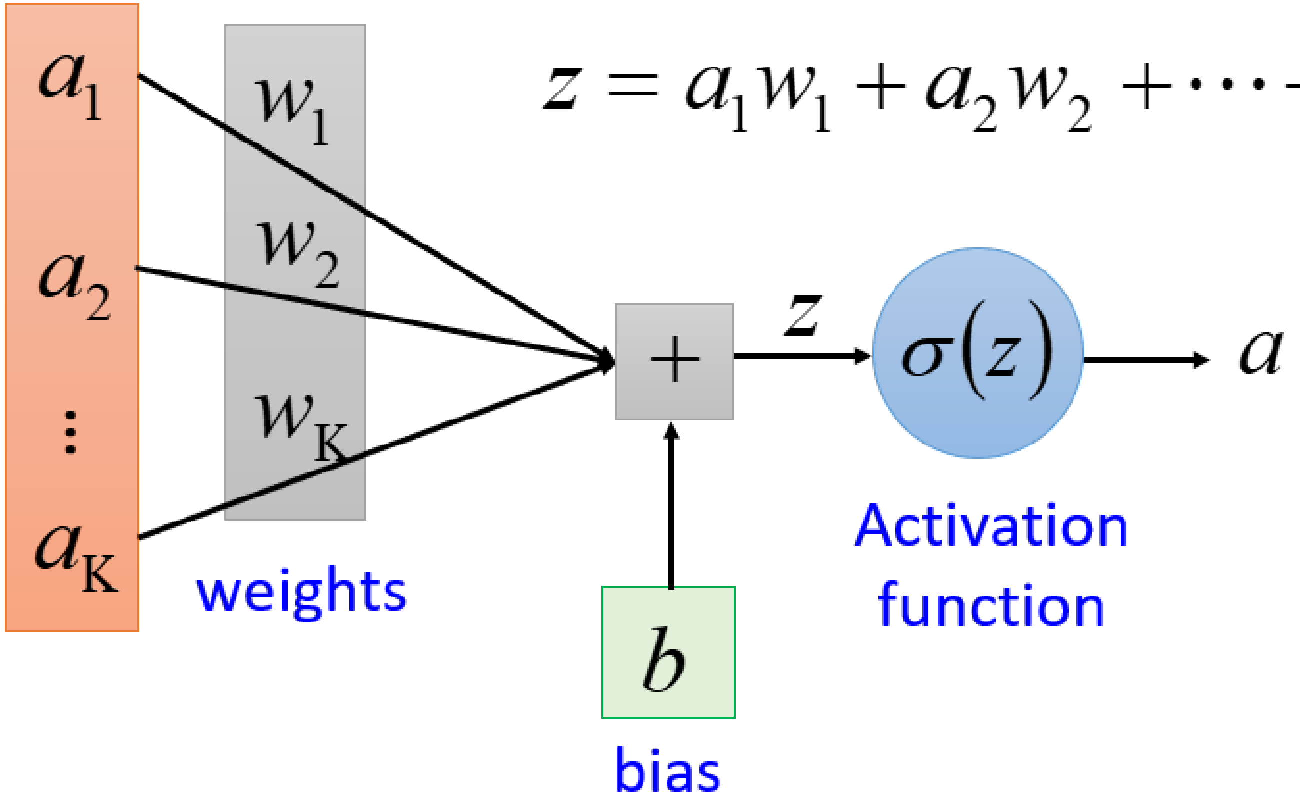
Deep Learning!

$$Q(s,a) = f_\theta(s,a)$$

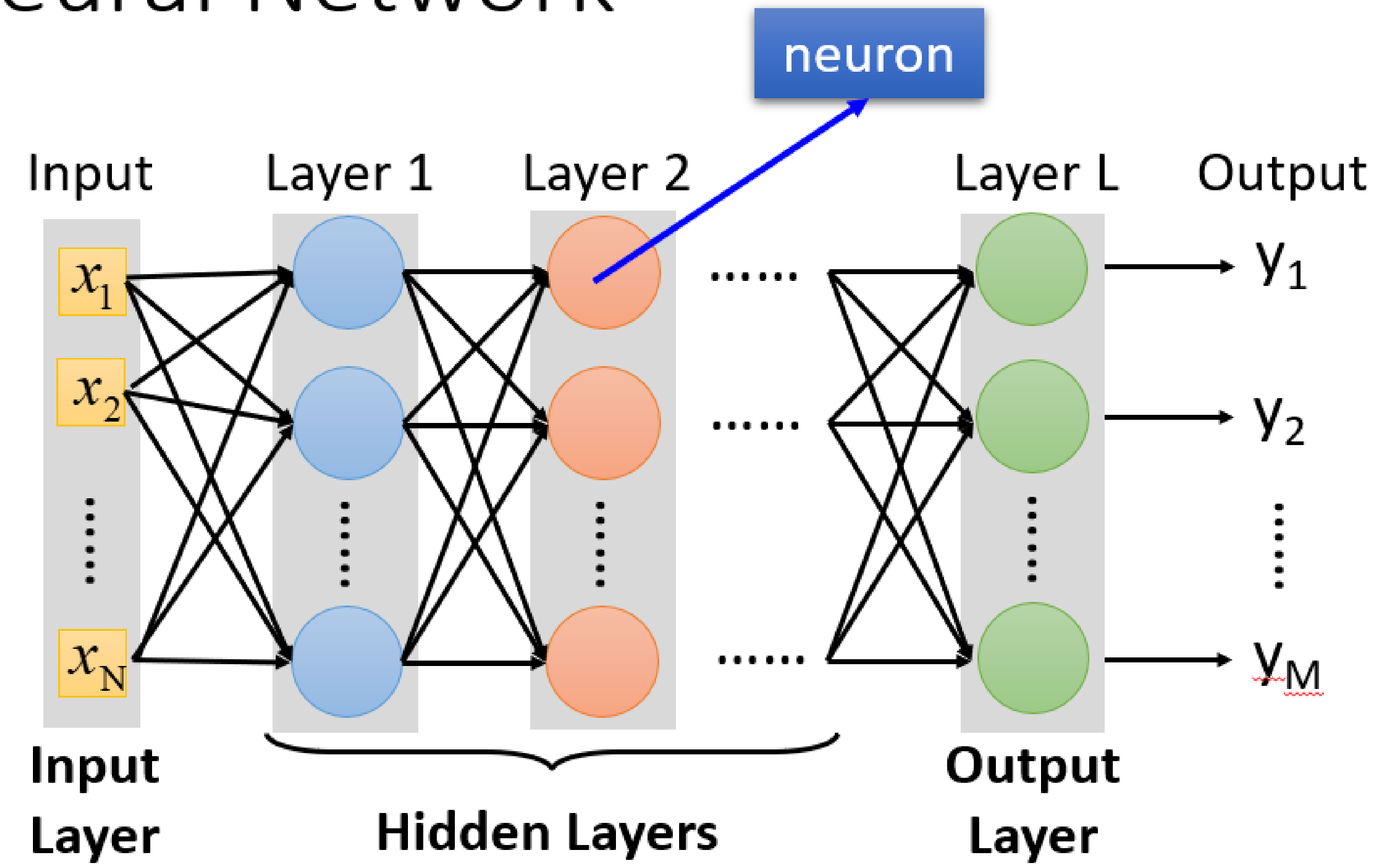
Element of Neural Network

Neuron $f: R^K \to R$

$$z = a_1 w_1 + a_2 w_2 + \cdots + a_K w_K + b$$

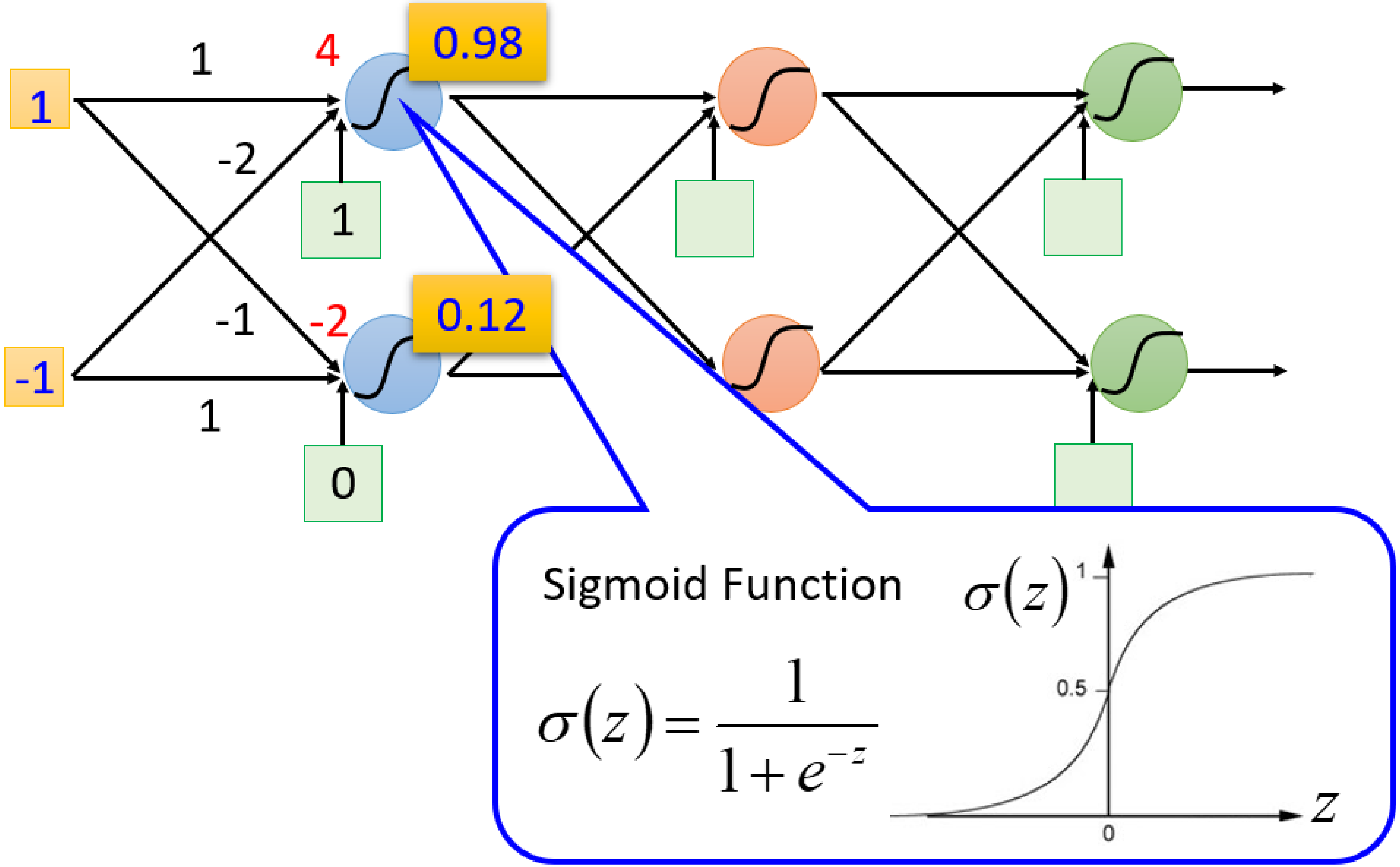


Neural Network



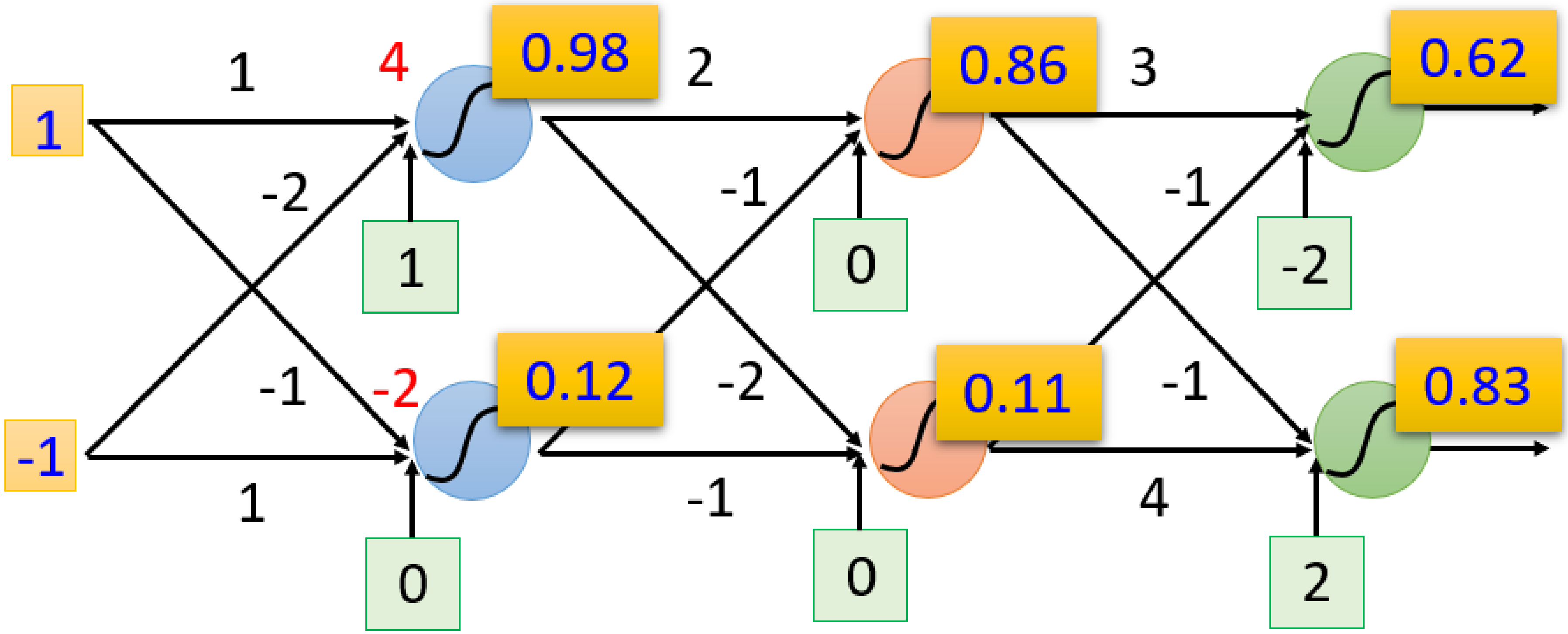
Deep means many hidden layers

Example of Neural Network



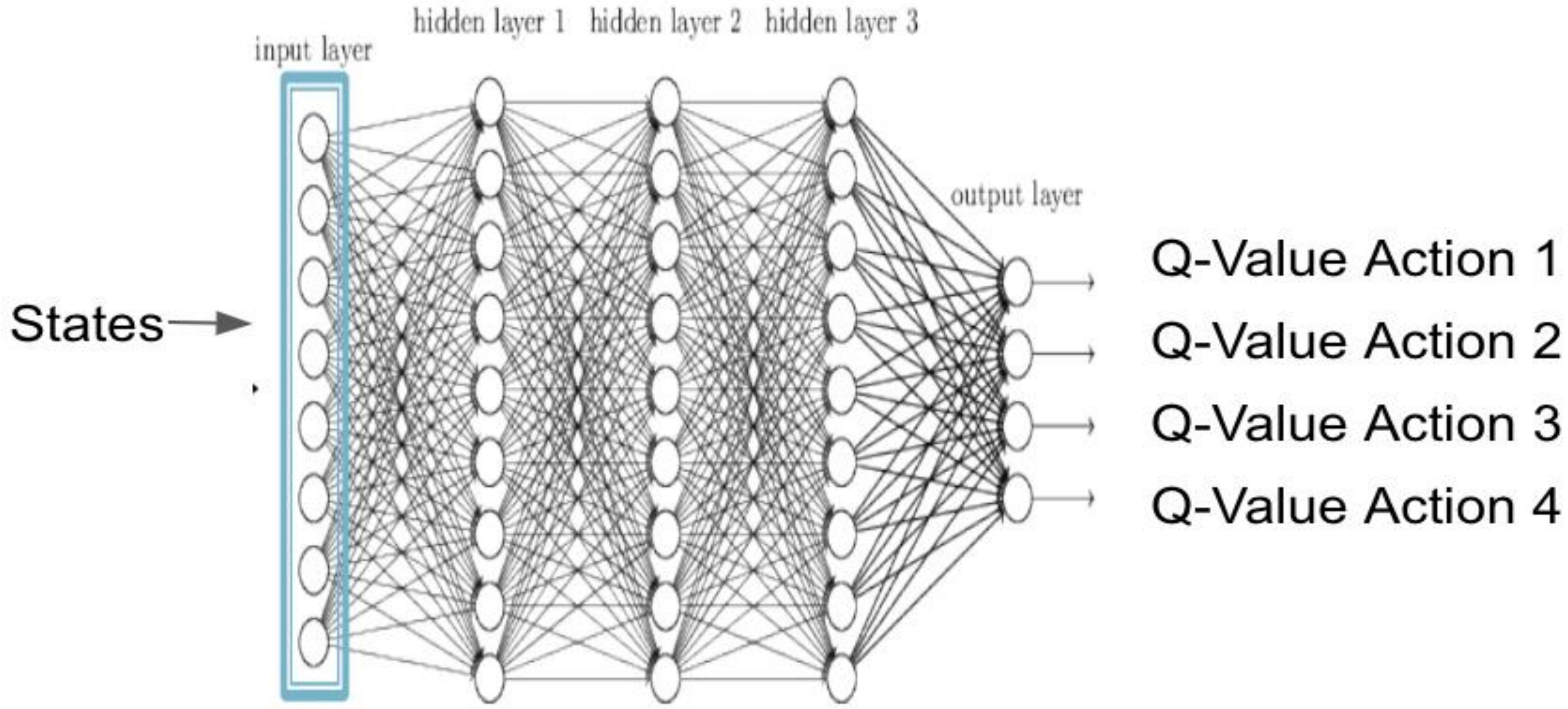
$$\sigma(z) = \frac{1}{1+e^{-z}}$$

Example of Neural Network



Changing the parameters (weights) changes the function!

Neural Networks: Non-linear function approximation



Differences between RL and Supervised Learning

Predicting State-Action Value

Input: (s,a)

Output: $Q_\theta(s,a)$

Target: $r + \gamma \max_{a'} Q_\theta(s', a')$

Predicting House Price

Input: size, #bedrooms, nearby school ratings, year built, etc.

Output: $f_\theta(\boldsymbol{x})$

Target: $\$680K$

RL has a non-stationary target! This leads to instabilities if using non-linear function approximation.

How to get Q-Learning to work with Deep Learning?

- Experience Replay Buffer
  - Don’t throw away each transition (s,a,r,s’)
  - Save them in a buffer or “replay memory”
  - During training randomly sample a batch of transitions to update Q

How to get Q-Learning to work with Deep Learning?

- Target Network
  - Keep the network for the target fixed and only update periodically

Like before we want to update Q to minimize the error:

$$error = \frac{1}{2}\left(r + \gamma \max_{a'} Q_T(s', a'; \theta^-) - Q(s, a; \theta)\right)^2$$

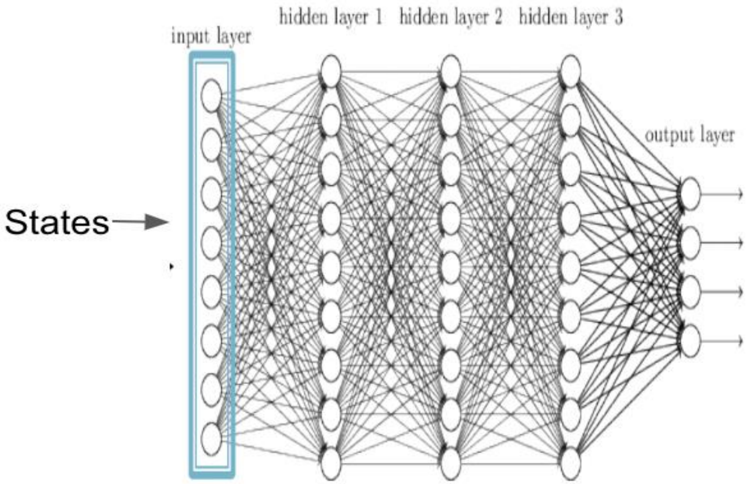
$$\nabla_\theta error = -\left(r + \gamma \max_{a'} Q_T(s', a'; \theta^-) - Q(s, a; \theta)\right) \nabla_\theta Q(s, a; \theta)$$

Take step to decrease error (in the direction of the negative gradient)

$$\theta \leftarrow \theta + \alpha\left(r + \gamma \max_{a'} Q_T(s', a'; \theta^-) - Q(s, a; \theta)\right) \nabla_\theta Q(s, a; \theta)$$

$$\theta \leftarrow \theta + \alpha \left(r + \gamma \max_{a'} Q_T(s', a'; \theta^-) - Q(s, a; \theta) \right) \nabla_\theta Q(s, a; \theta)$$

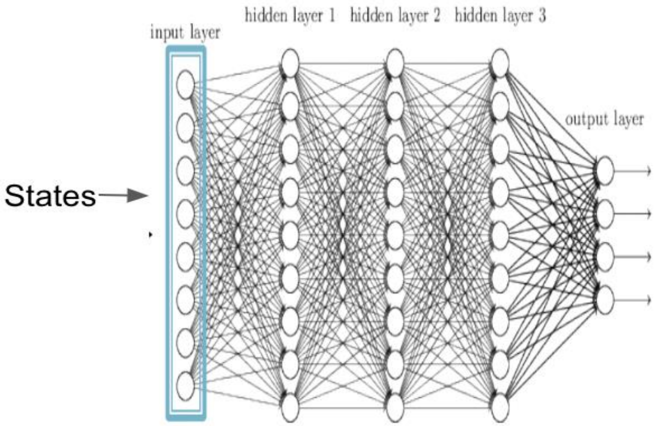
Q Network



$Q(s, a; \theta)$

Updates θ^- every C timesteps

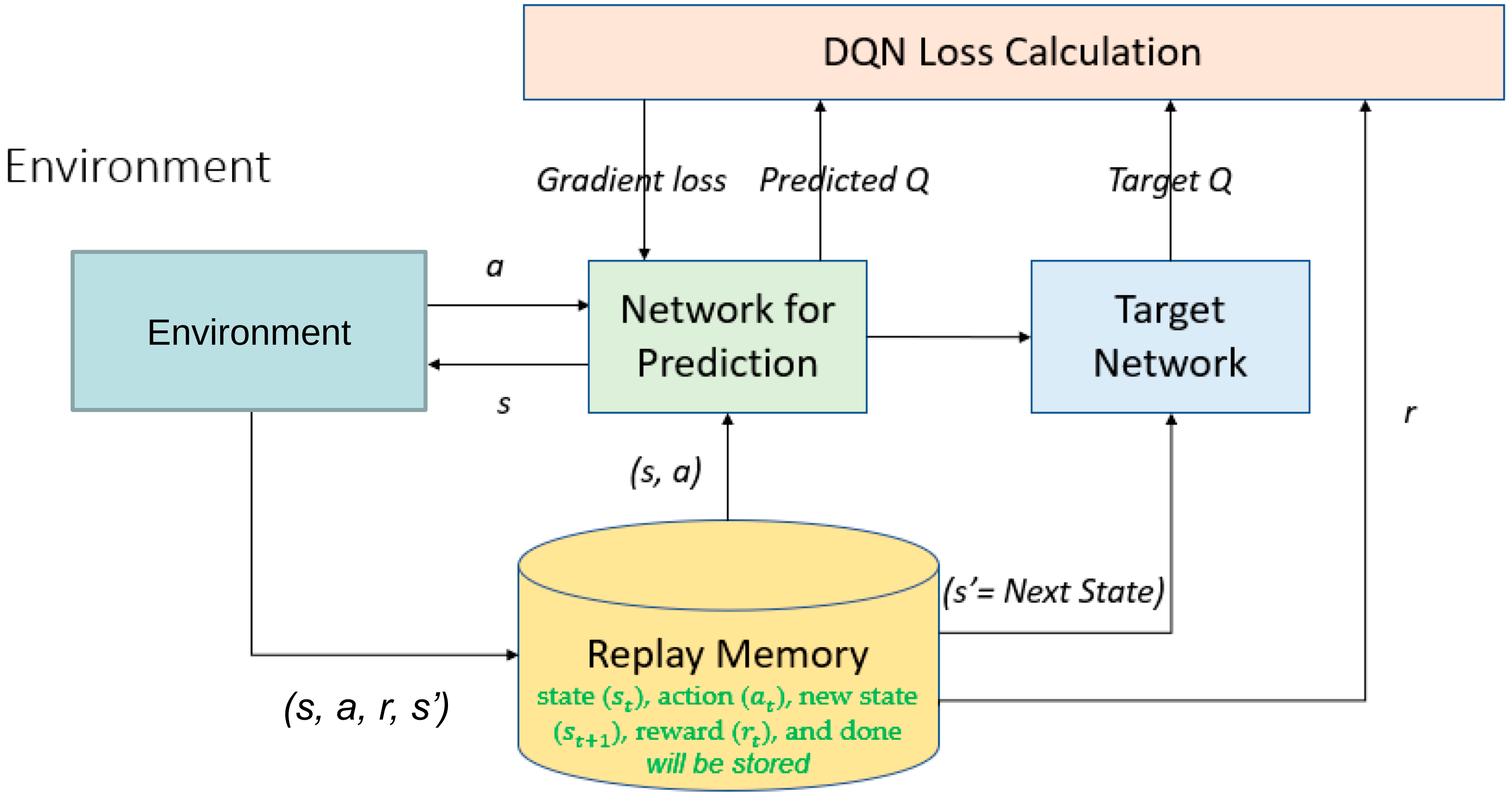
Target Network



$Q_T(s', a'; \theta^-)$

High-Level Overview of DQN

$$\theta \leftarrow \theta + \alpha\left(r + \gamma \max_{a'} Q_T(s', a'; \theta^-) - Q(s, a; \theta)\right) \nabla_\theta Q(s, a; \theta)$$



Deep RL Makes a Big Splash!

nature

Explore content ⌄ About the journal ⌄ Publish with us ⌄ Subscribe

nature › letters › article

Published: 25 February 2015

Human-level control through deep reinforcement learning

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg & Demis Hassabis

Startups

Google Acquires Artificial Intelligence Startup DeepMind For More Than $500M

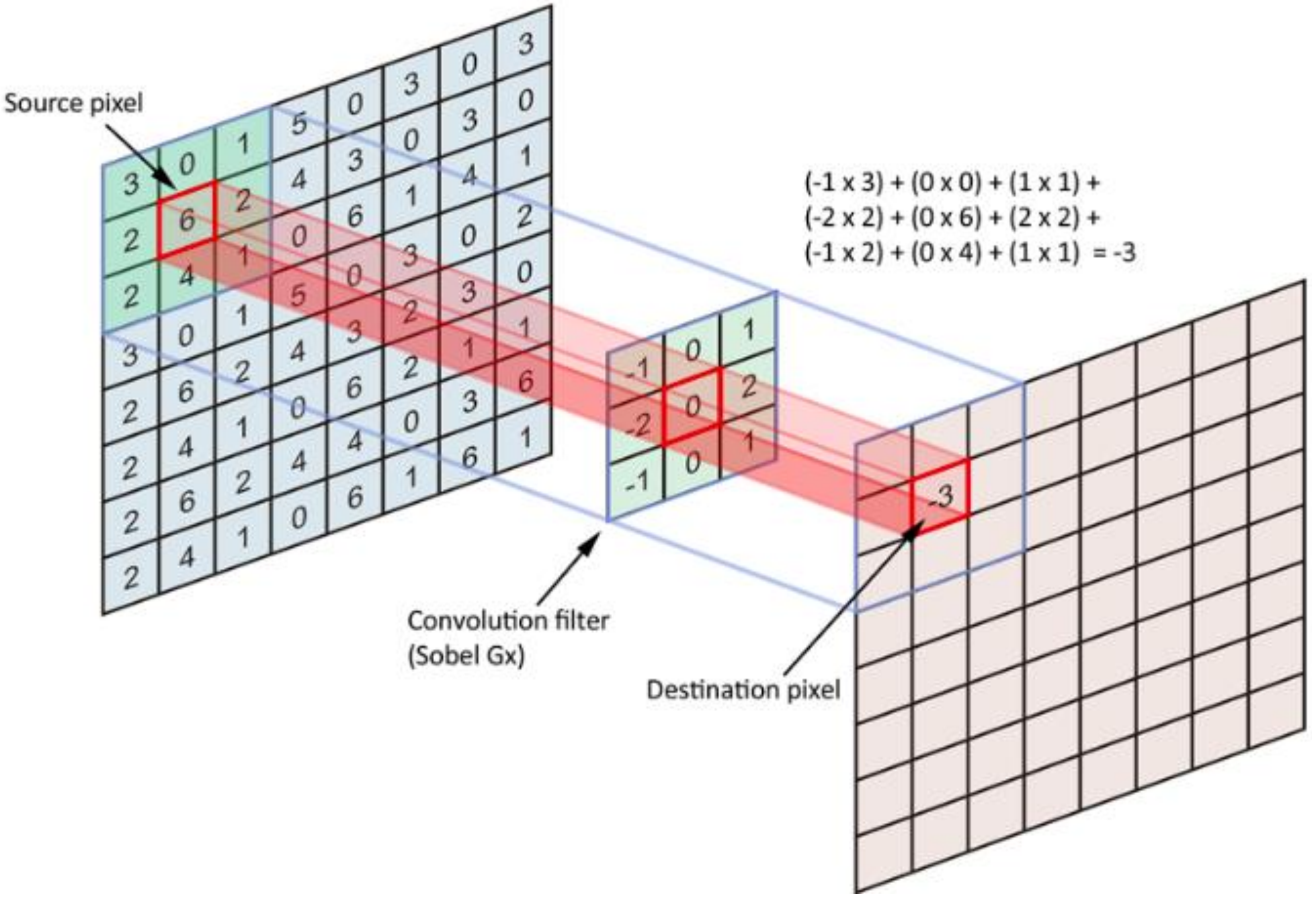
Catherine Shu @catherineshu / 6:20 PM MST • January 26, 2014

Comment

The Arcade Learning Environment

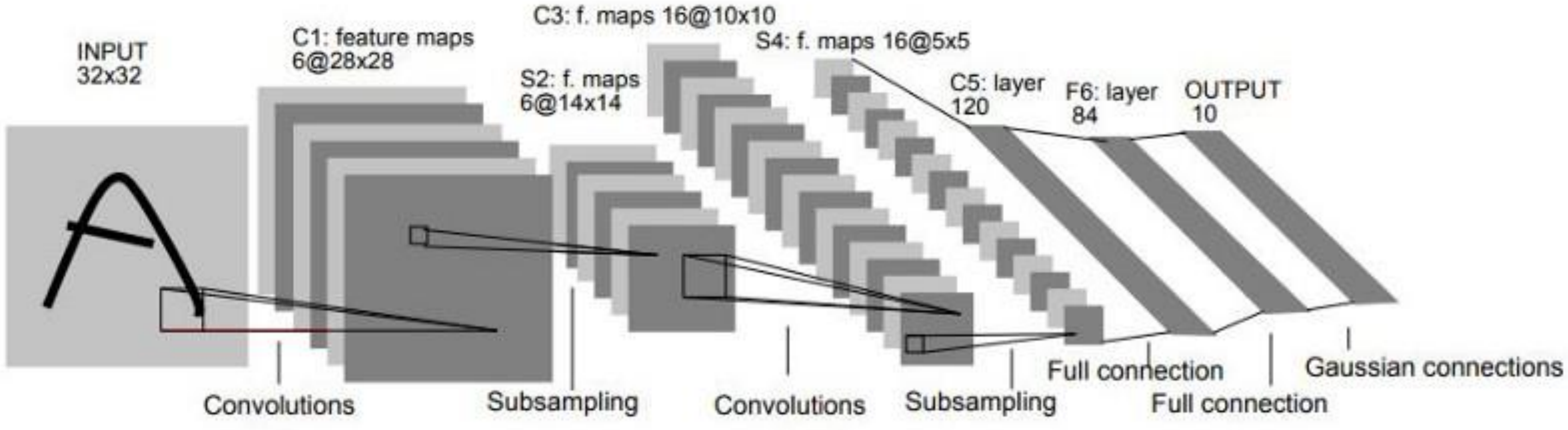
How do you learn from raw pixels?

- Too many parameters to have a weight for each pixel.
- Use a convolutional filter



How do you learn from raw pixels?

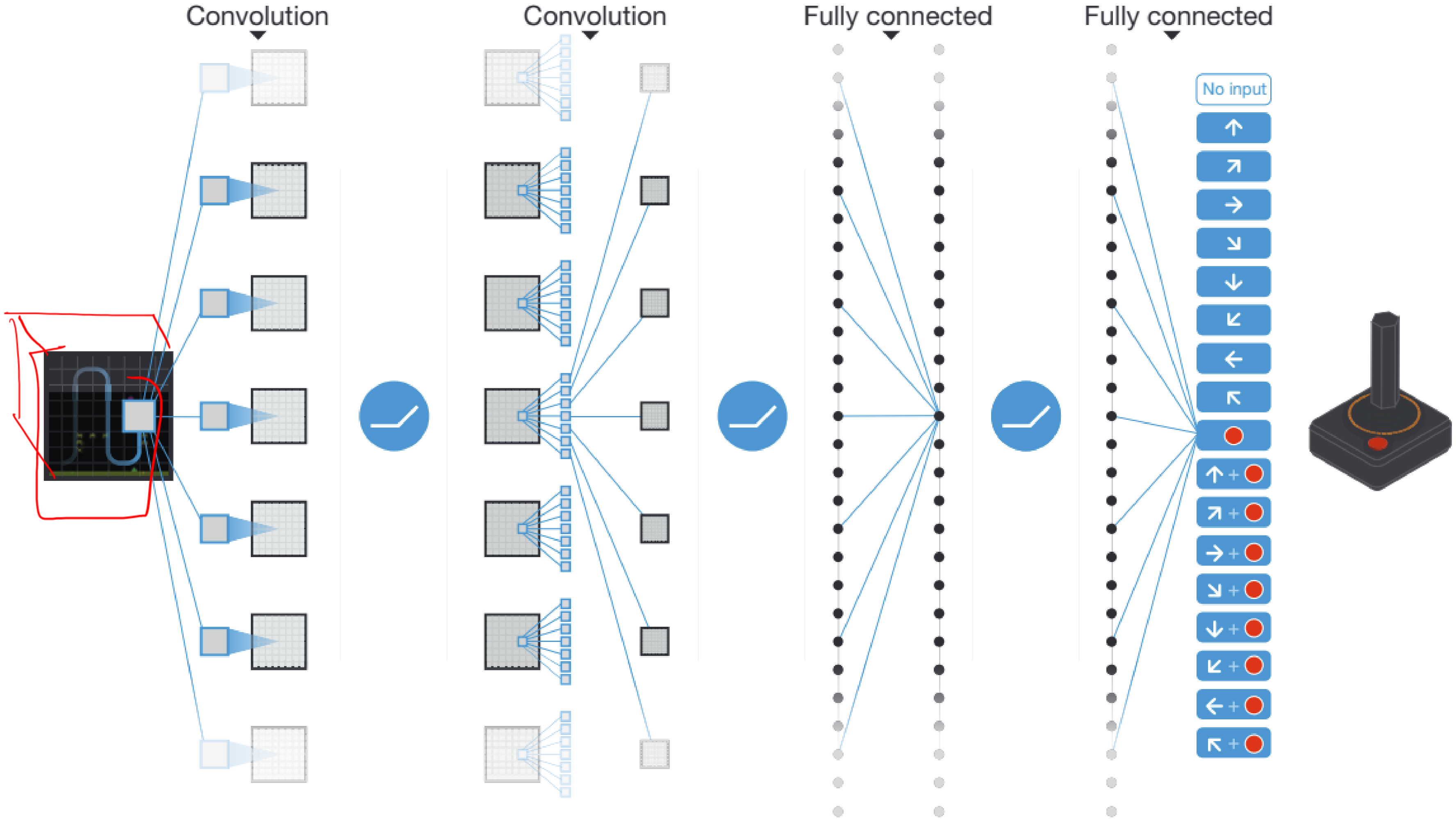
- Too many parameters to have a weight for each pixel.
- Use a convolutional filter
- Use several layers of multiple filters



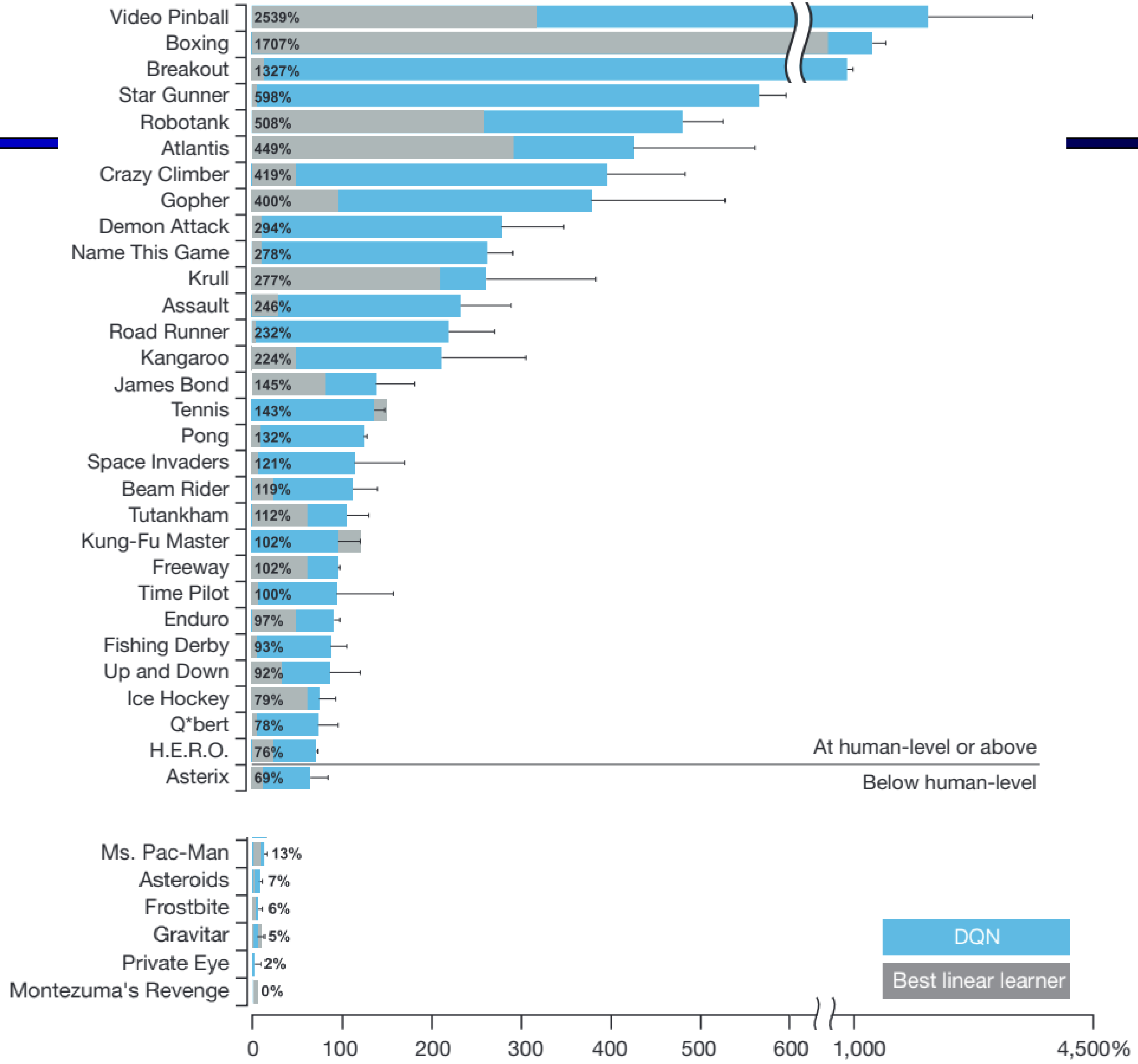
LeCun, Yann, et al. "Gradient-based learning applied to document recognition." 1998.

High-Level Architecture

- Learns to "see" through trial and error!
- Learns what actions to take to maximize game score.
- Epsilon-greedy exploration.




021 3 1



Video Pinball 2539%
Boxing 1707%
Breakout 1327%
Star Gunner 598%
Robotank 508%
Atlantis 449%
Crazy Climber 419%
Gopher 400%
Demon Attack 294%
Name This Game 278%
Krull 277%
Assault 246%
Road Runner 232%
Kangaroo 224%
James Bond 145%
Tennis 143%
Pong 132%
Space Invaders 121%
Beam Rider 119%
Tutankham 112%
Kung-Fu Master 102%
Freeway 102%
Time Pilot 100%
Enduro 97%
Fishing Derby 93%
Up and Down 92%
Ice Hockey 79%
Q*bert 78%
H.E.R.O. 76%
Asterix 69%
At human-level or above
Below human-level
Ms. Pac-Man 13%
Asteroids 7%
Frostbite 6%
Gravitar 5%
Private Eye 2%
Montezuma's Revenge 0%
DQN
Best linear learner
0
100
200
300
400
500
600
1,000
4,500%

Lots of Advanced Exploration Strategies

Unifying Count-Based Exploration and Intrinsic Motivation

Marc G. Bellemare bellemare@google.com
Sriram Srinivasan srsrinivasan@google.com
Georg Ostrovski ostrovski@google.com
Tom Schaul schaul@google.com
David Saxton saxton@google.com
Rémi Munos munos@google.com
Google DeepMind
London, United Kingdom

INCENTIVIZING EXPLORATION IN REINFORCEMENT LEARNING WITH DEEP PREDICTIVE MODELS

Bradly C. Stadie
Department of Statistics
University of California, Berkeley
Berkeley, CA 94720
bstadie@berkeley.edu

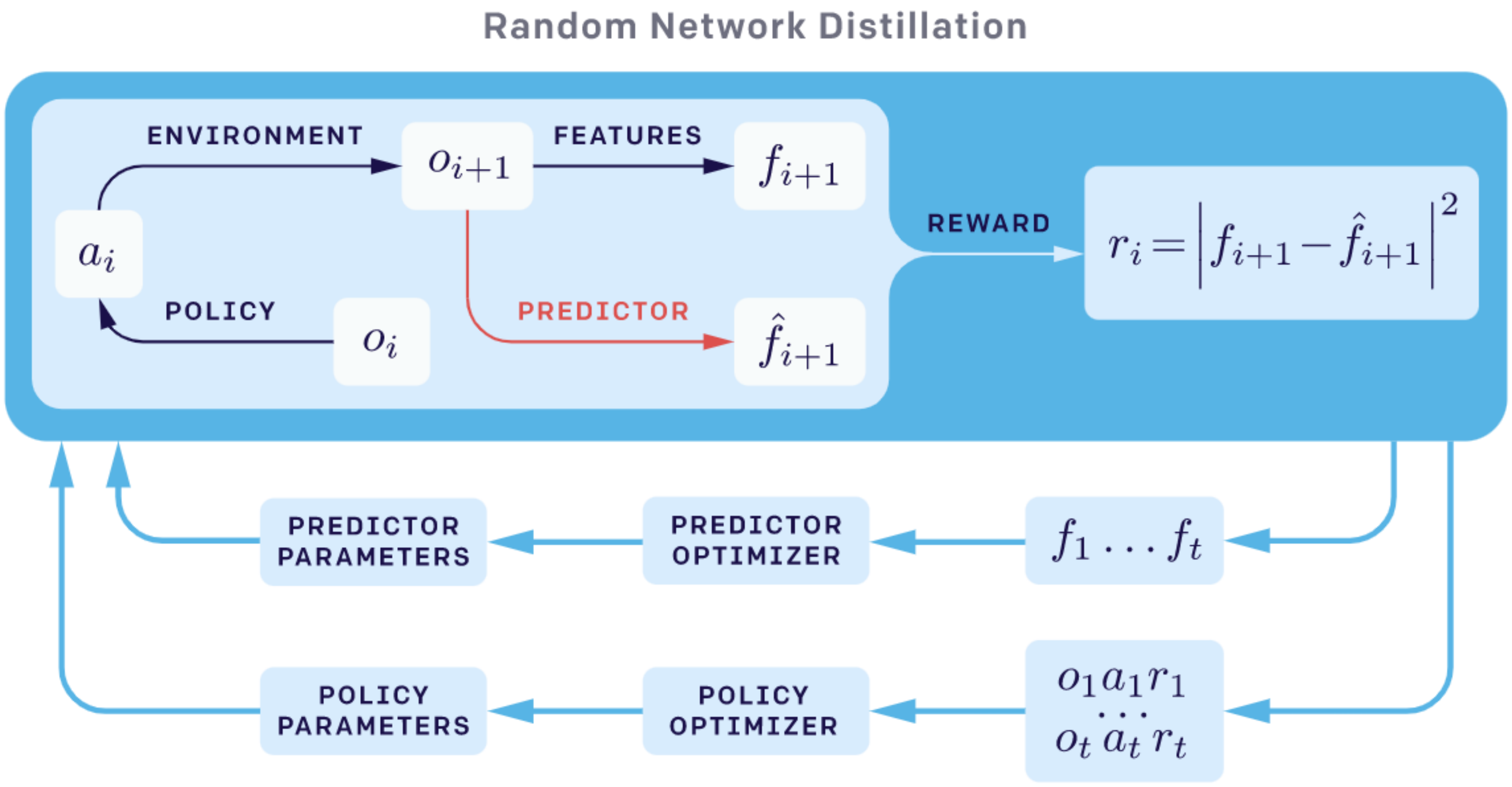
Sergey Levine **Pieter Abbeel**
EECS Department
University of California, Berkeley
Berkeley, CA 94720
{svlevine,pabbeel}@cs.berkeley.edu

EXPLORATION BY RANDOM NETWORK DISTILLATION

Yuri Burda* OpenAI
Harrison Edwards* OpenAI
Amos Storkey Univ. of Edinburgh
Oleg Klimov OpenAI

Great blog article: https://lilianweng.github.io/posts/2020-06-07-exploration-drl/

Exploration by Random Network Distillation



DQN only works for discrete action spaces

- Next Time: How to deal with continuous action spaces