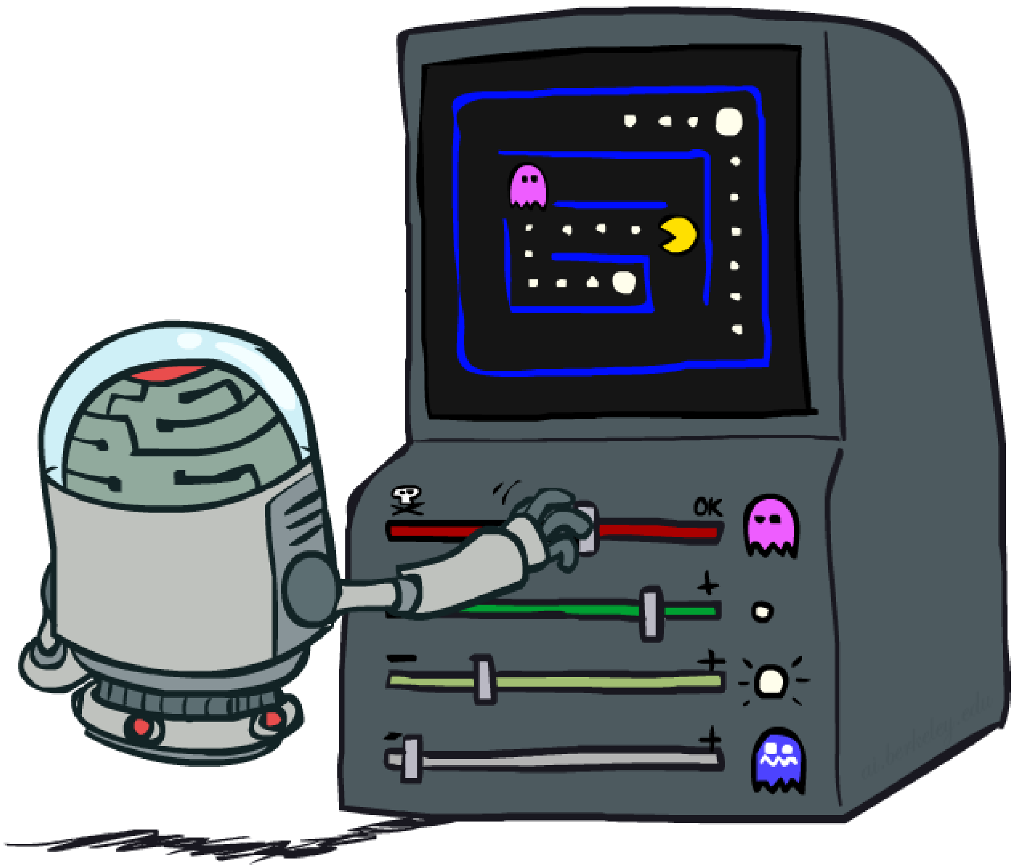
CS 6300: Artificial Intelligence

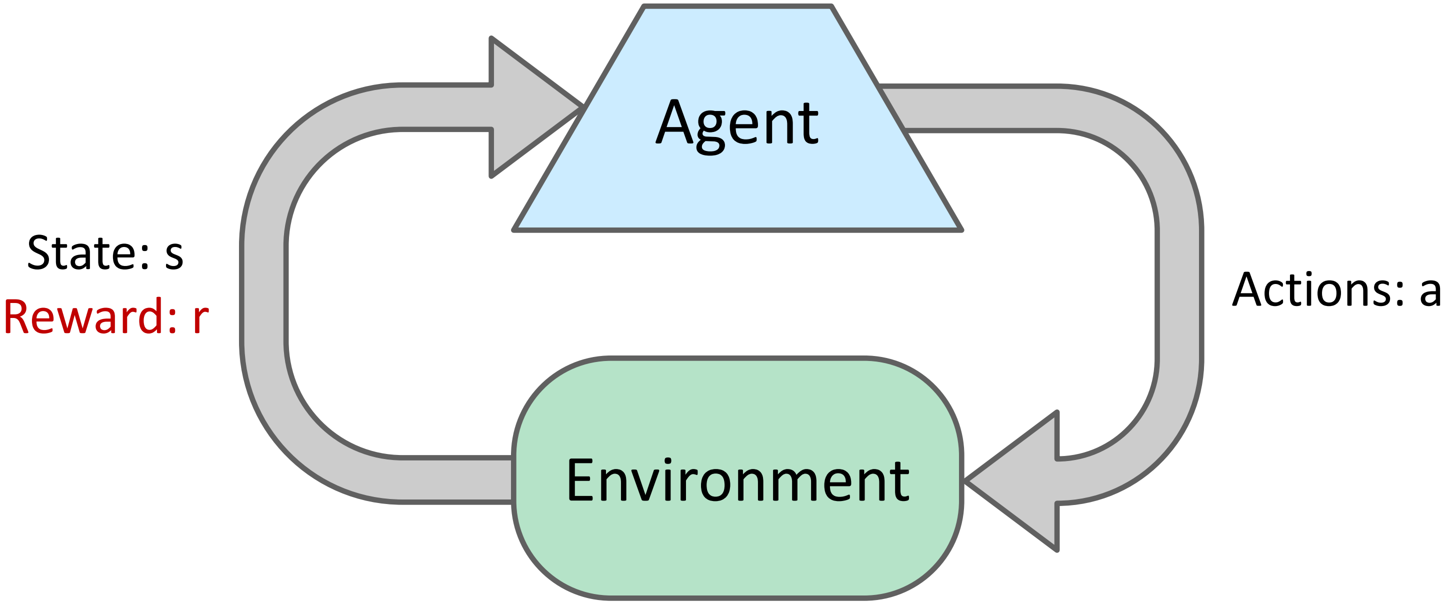
Reinforcement Learning II: Function Approximation



Instructor: Daniel Brown --- University of Utah

[Based on slides created by Dan Klein and Pieter Abbeel http://ai.berkeley.edu.]

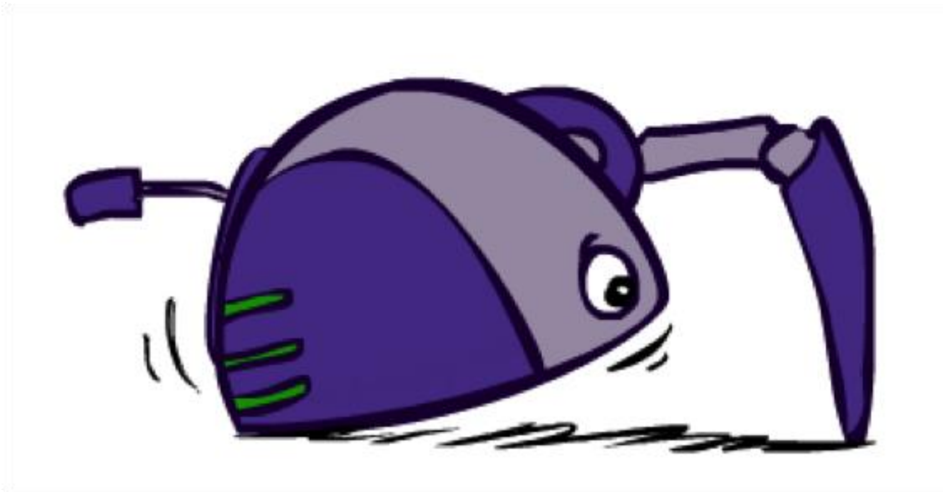
Reinforcement Learning



- 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

- We still assume an 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, so must try out actions

- Big idea: Compute all averages over T using sample outcomes

The Story So Far: MDPs and RL

Known MDP: Offline Solution

Goal	Technique
Compute V*, Q*, π*	Value / policy iteration
Evaluate a fixed policy π	Policy evaluation

Unknown MDP: Model-Based

Goal	Technique
Compute V*, Q*, π*	VI/PI on approx. MDP
Evaluate a fixed policy π	PE on approx. MDP

Unknown MDP: Model-Free

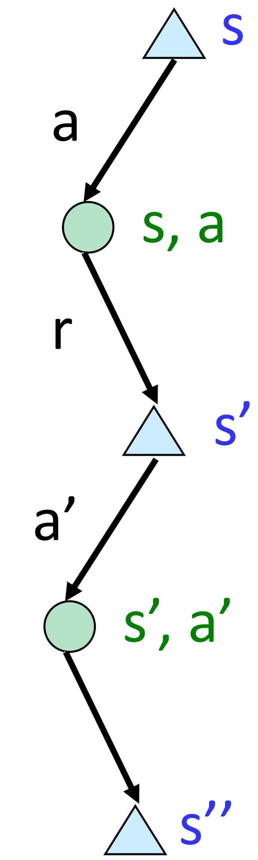
Goal	Technique
Compute V*, Q*, π*	Q-learning
Evaluate a fixed policy π	Value Learning

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

- 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

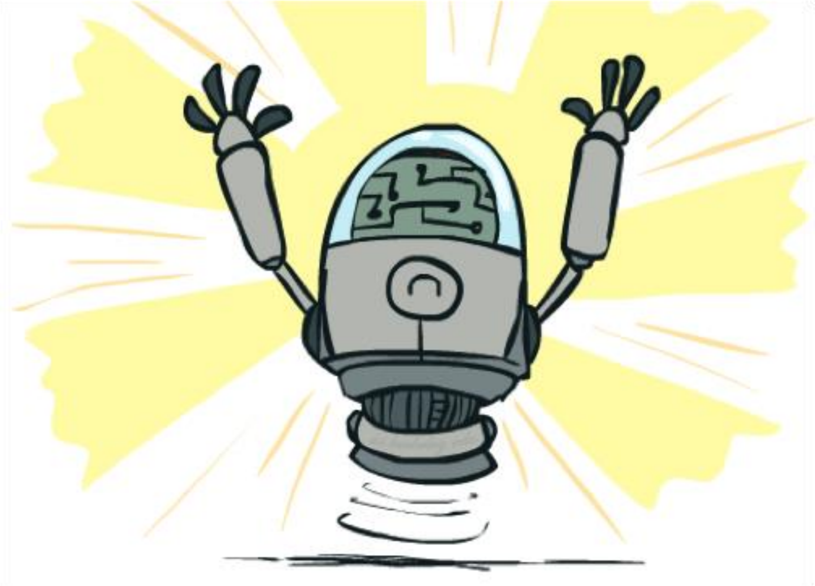
$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\left[r + \gamma \max_{a'} Q(s',a') \right]$$

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

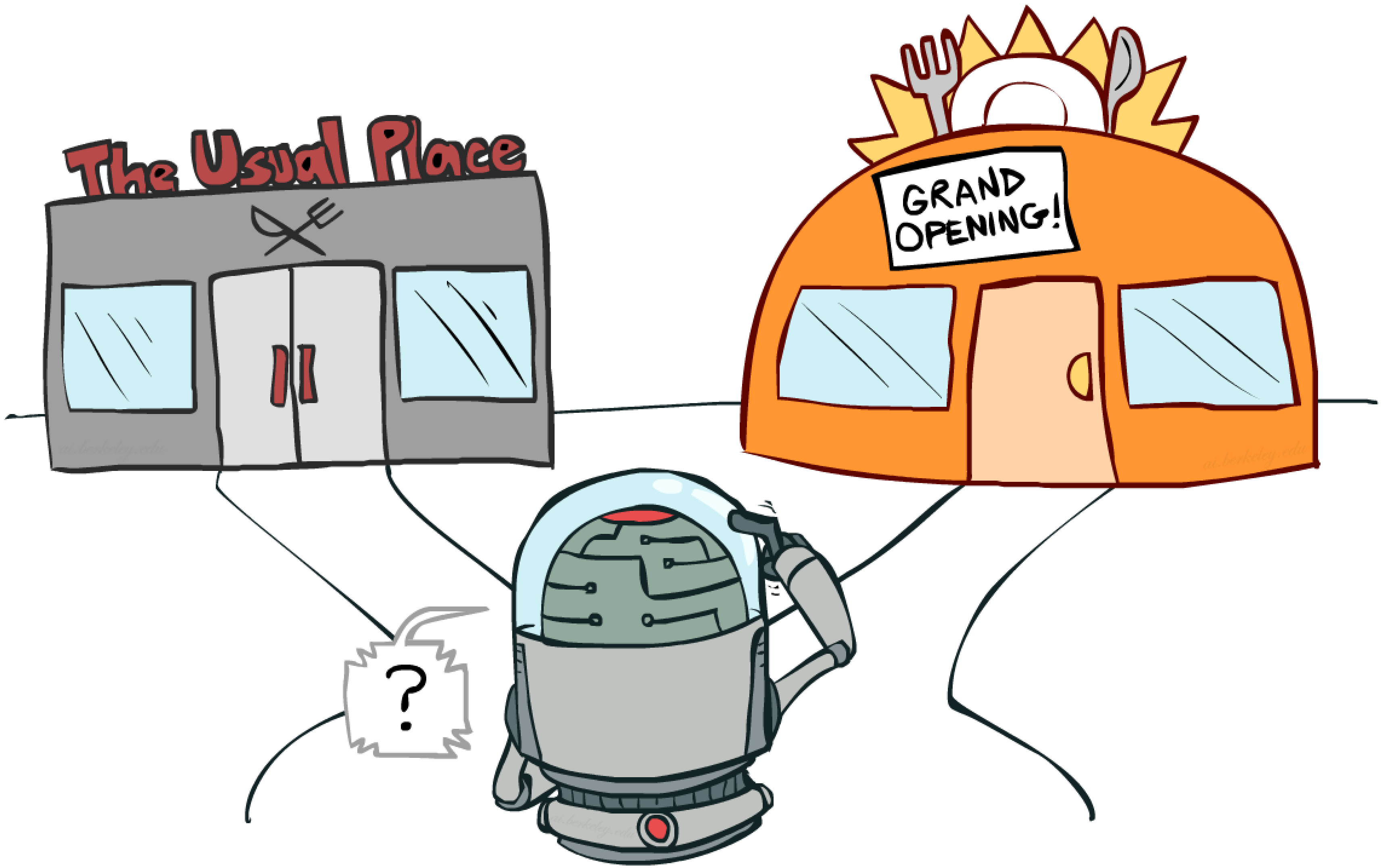
Useful alternate form of update for Q-learning. We want to push the Q-value towards the sample!

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 (!)



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. $f(u,n) = u + k/n$

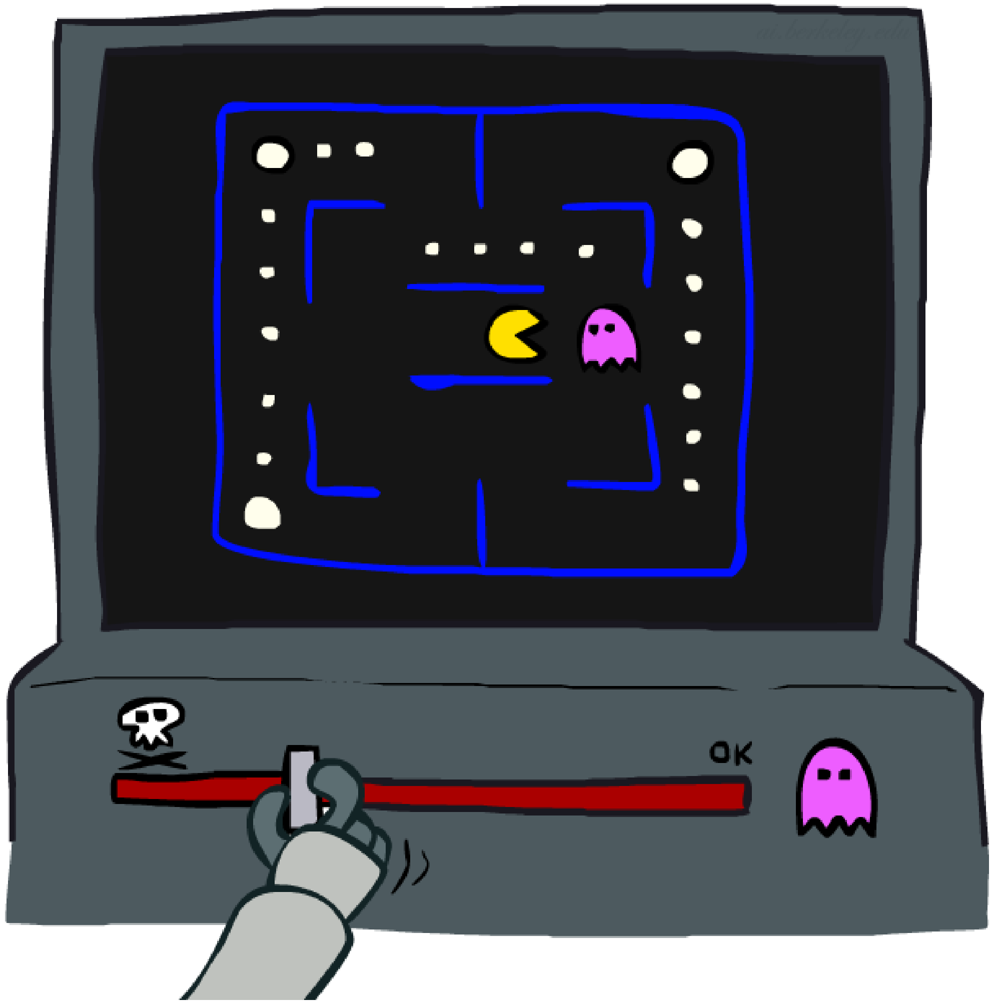
Regular Q-Update: $Q(s,a) \leftarrow_\alpha R(s,a,s') + \gamma \max_{a'} Q(s',a')$

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!

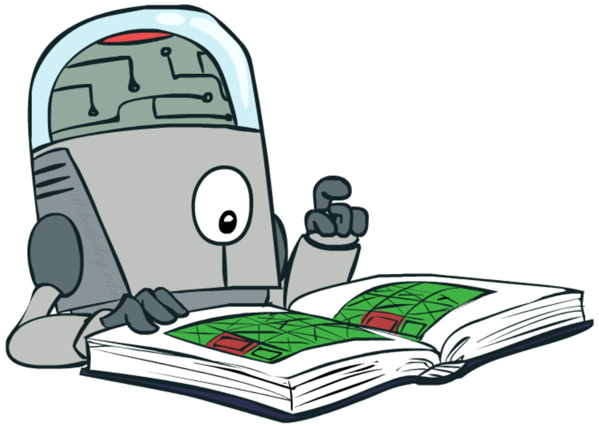
[Demo: exploration – Q-learning – crawler – exploration function (L11D4)]

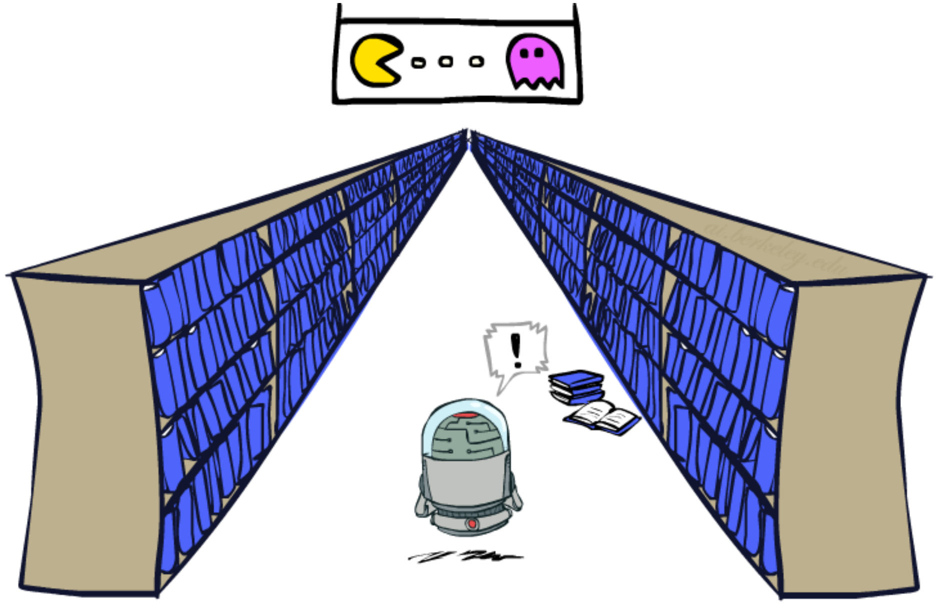
Approximate Q-Learning



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

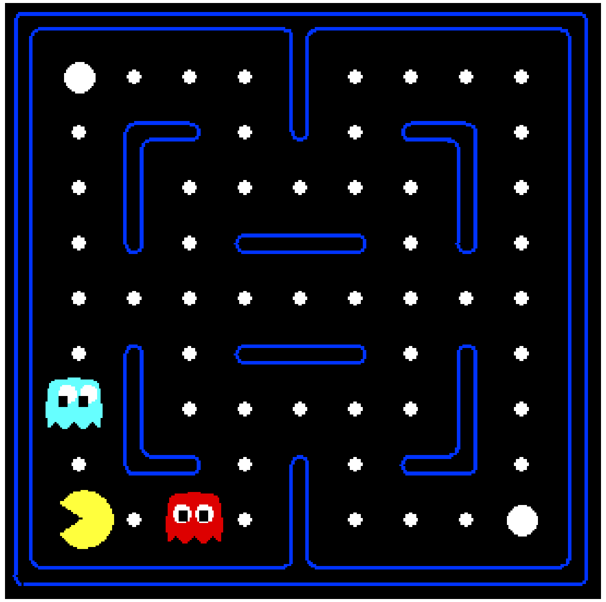




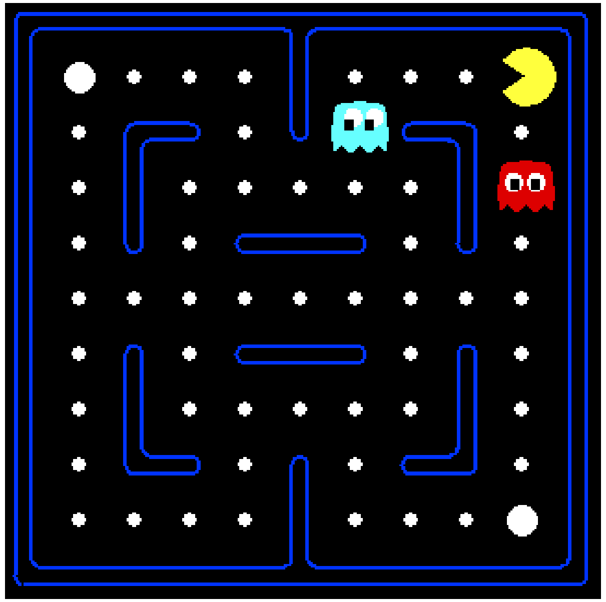
[demo – RL pacman]

Example: Pacman

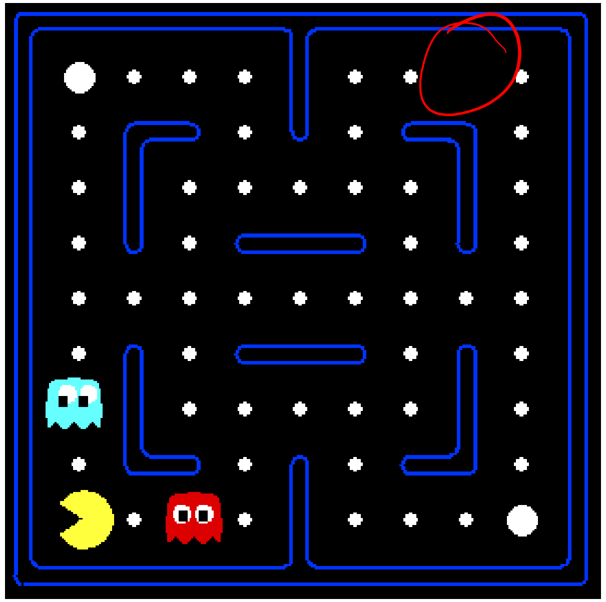
Let's say we discover through experience that this state is bad:



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

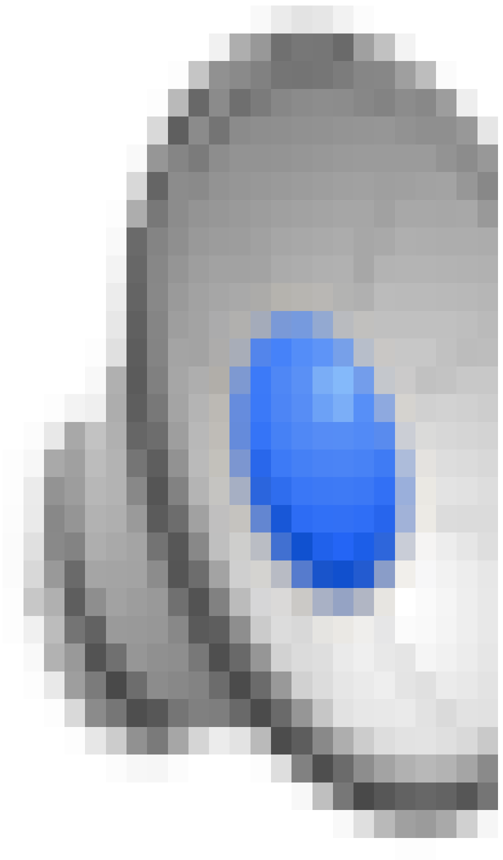


Or even this one!



[Demo: Q-learning – pacman – tiny – watch all (L11D5)]
[Demo: Q-learning – pacman – tiny – silent train (L11D6)]
[Demo: Q-learning – pacman – tricky – watch all (L11D7)]

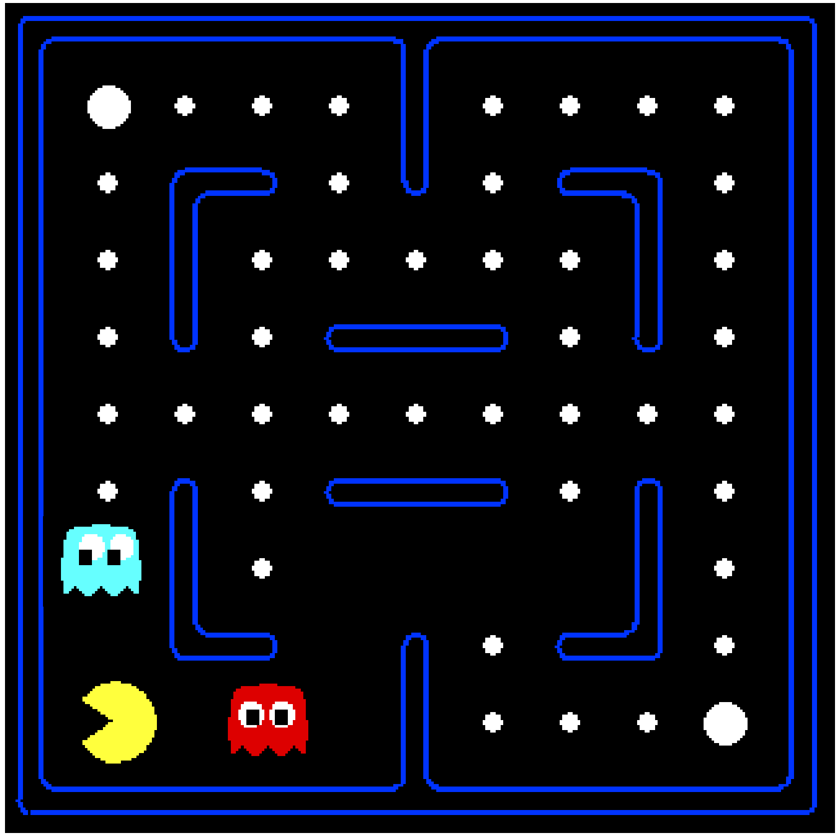
Video of Demo Q-Learning Pacman – Tiny – Watch All



Video of Demo Q-Learning Pacman – Tiny – Silent Train

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

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

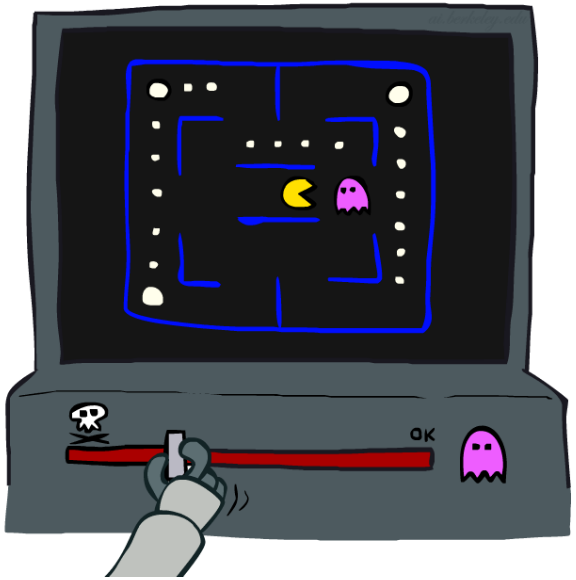
- Q-learning with linear Q-functions:

  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



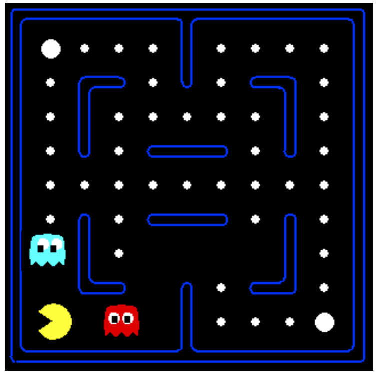
- Intuitive interpretation:
  - Adjust weights of active features
  - E.g., if something unexpectedly bad happens, blame the features that were on: disprefer all states with that state's features

- Formal justification: online least squares

Example: Q-Pacman

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

s

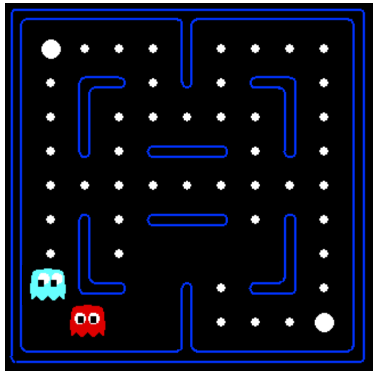


$f_{DOT}(s, \text{NORTH}) = 0.5$

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

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

s'



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

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

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

difference $= -501$

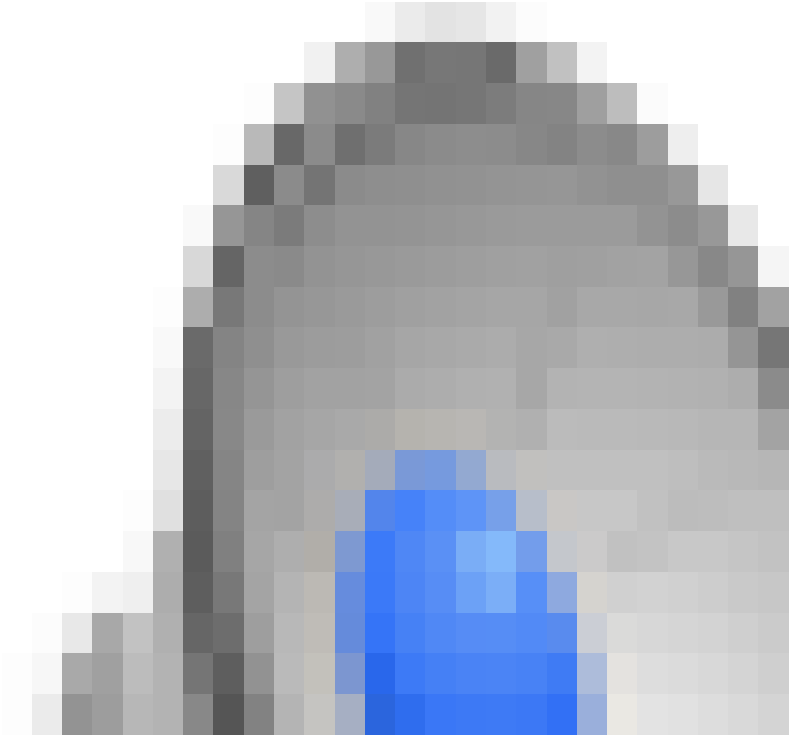
$w_{DOT} \leftarrow 4.0 + \alpha \left[-501\right] 0.5$

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

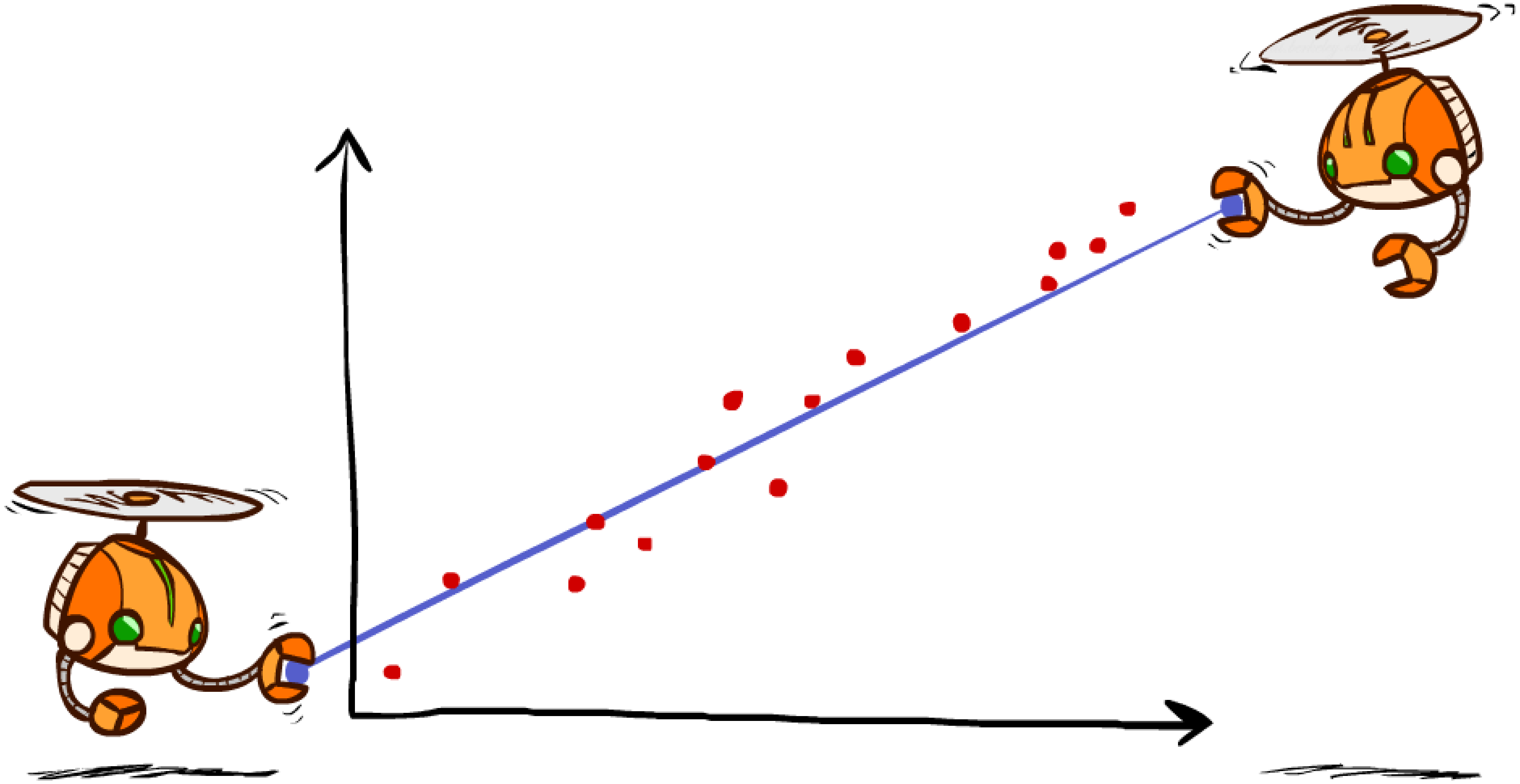
$$Q(s,a) = 3.0 f_{DOT}(s,a) - 3.0 f_{GST}(s,a)$$

[Demo: approximate Q-learning pacman (L11D10)]

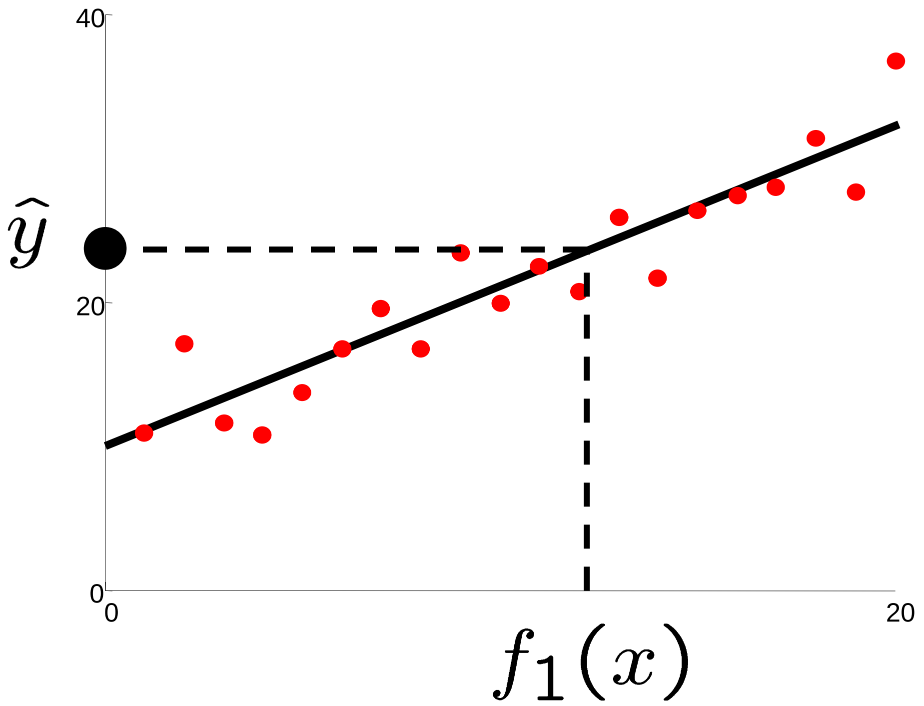
Video of Demo Approximate Q-Learning -- Pacman

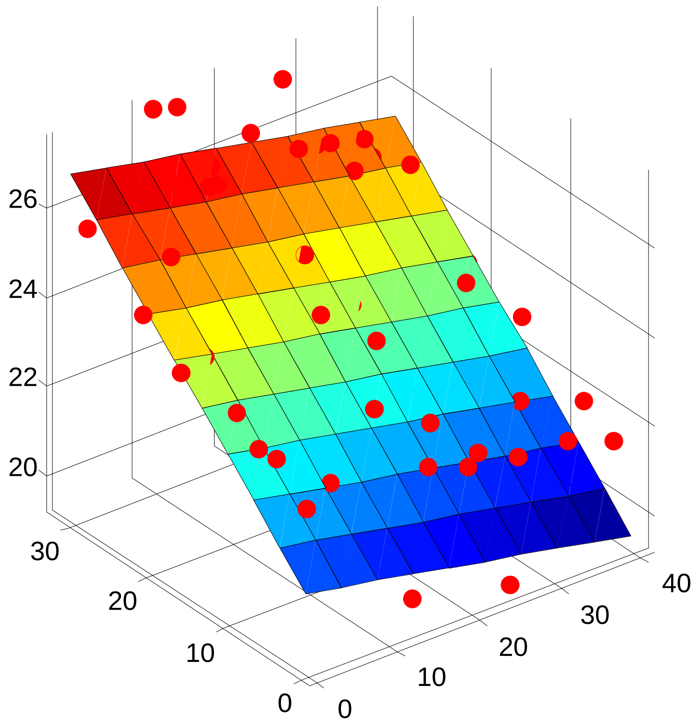


Q-Learning and Least Squares



Linear Approximation: Regression





Prediction:

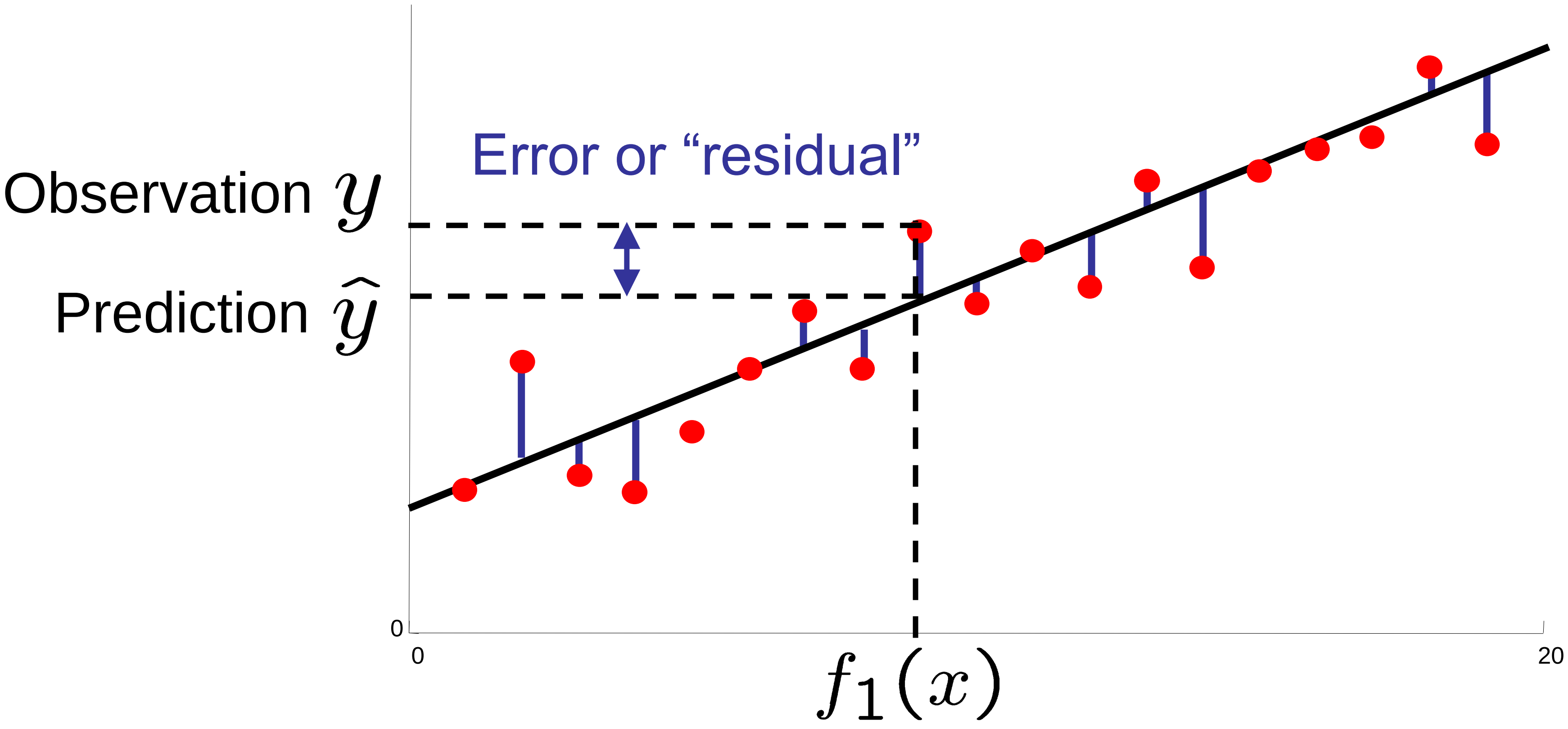
$$\hat{y} = w_0 + w_1 f_1(x)$$

Prediction:

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

Optimization: Least Squares

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



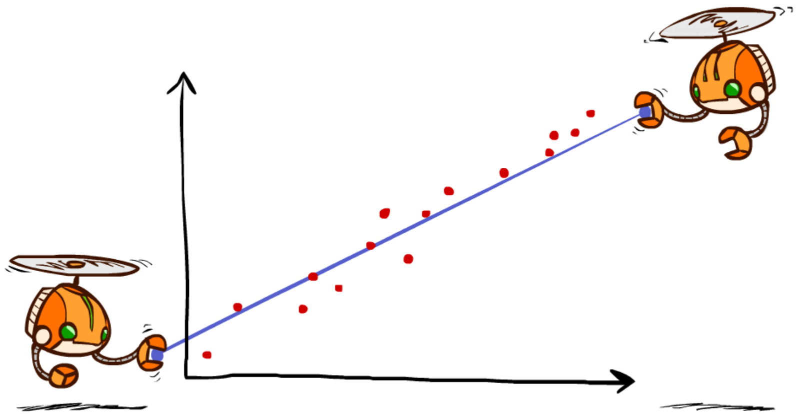
Minimizing Error

Imagine we had only one point x, with features f(x), target value y, and weights w:

$$\text{error}(w) = \frac{1}{2}\left(y - \sum_k w_k f_k(x)\right)^2$$

$$\frac{\partial\, \text{error}(w)}{\partial w_m} = -\left(y - \sum_k w_k f_k(x)\right) f_m(x)$$

$$w_m \leftarrow w_m + \alpha \left(y - \sum_k w_k f_k(x)\right) f_m(x)$$



Approximate q update explained:

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

“target” “prediction”

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.

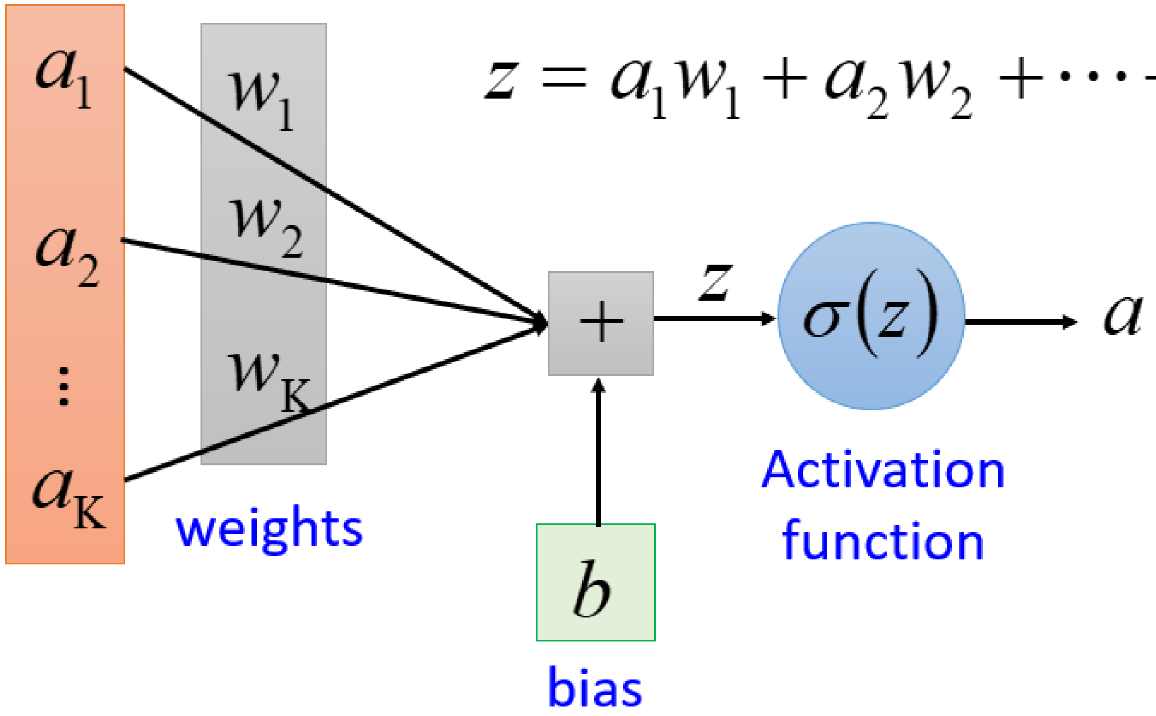
$$V(s) = f_\theta(s)$$

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

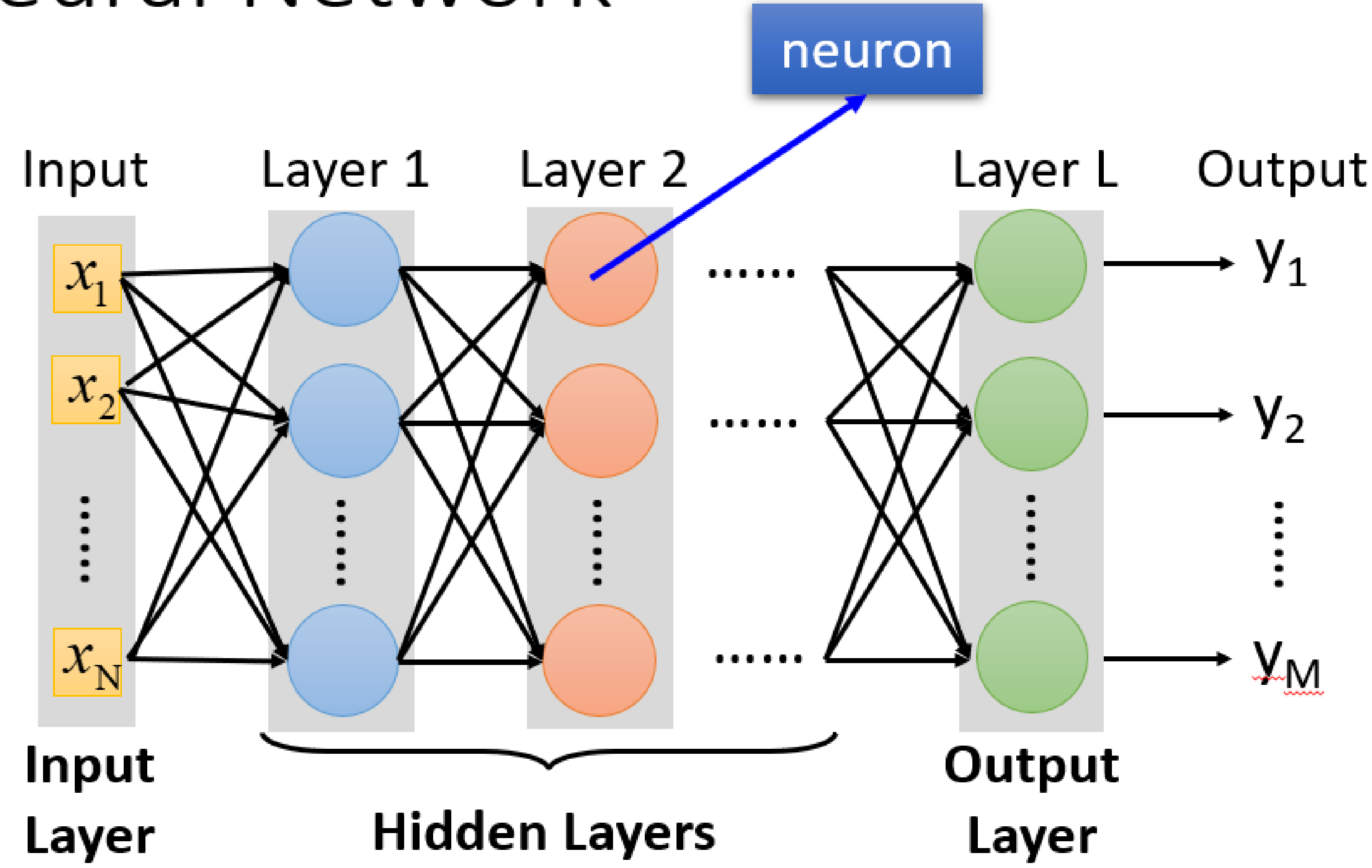
Element of Neural Network

Neuron $f: R^K \rightarrow R$

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

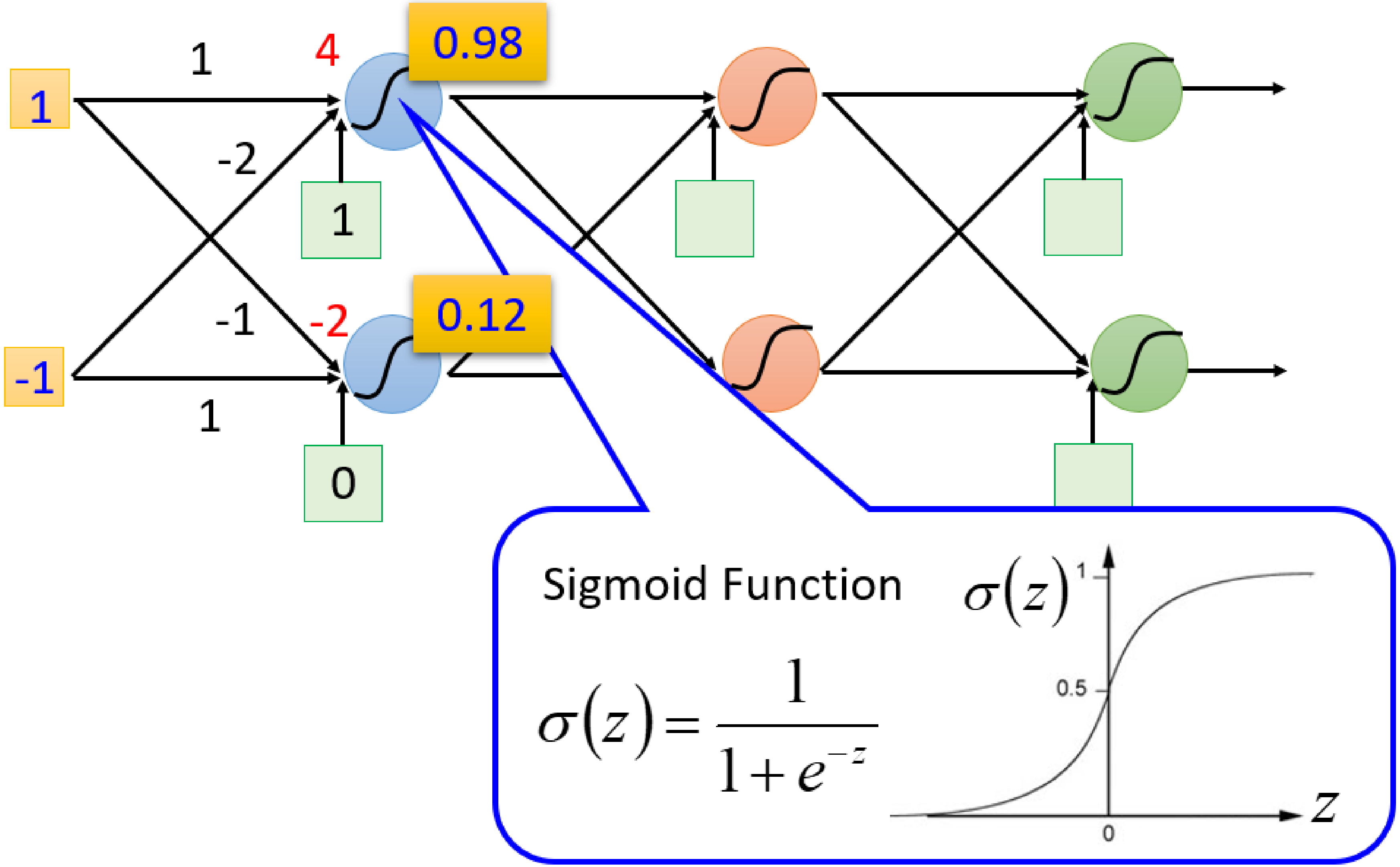


Neural Network

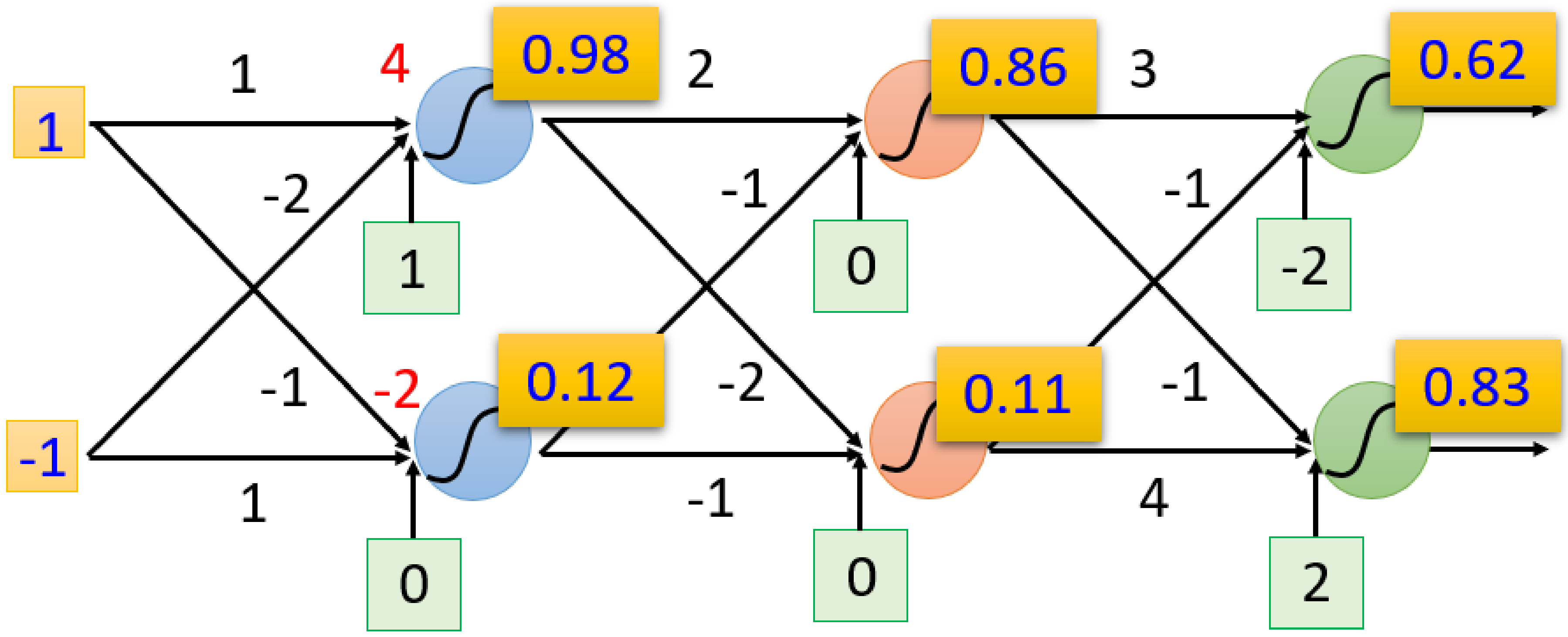


Deep means many hidden layers

Example of Neural Network

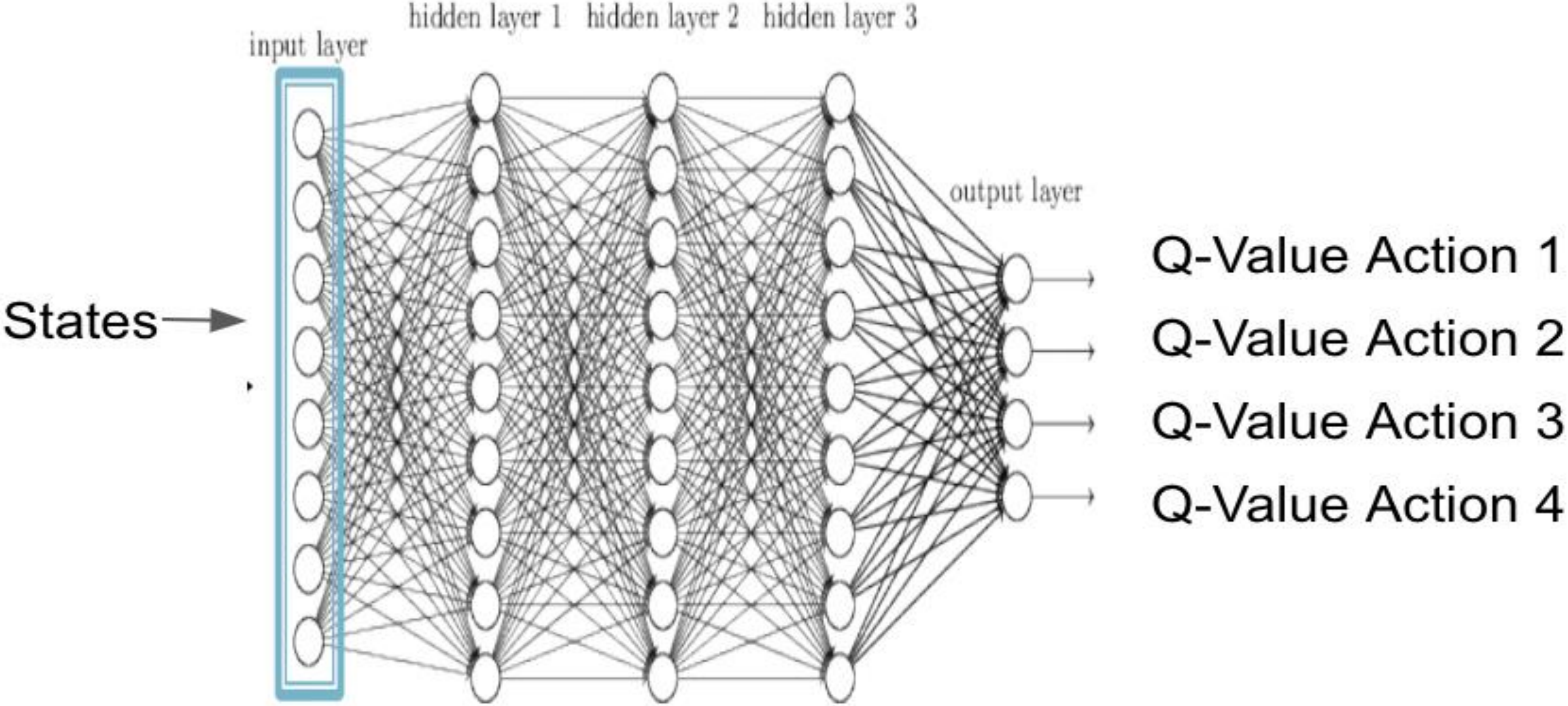


Example of Neural Network



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:

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

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

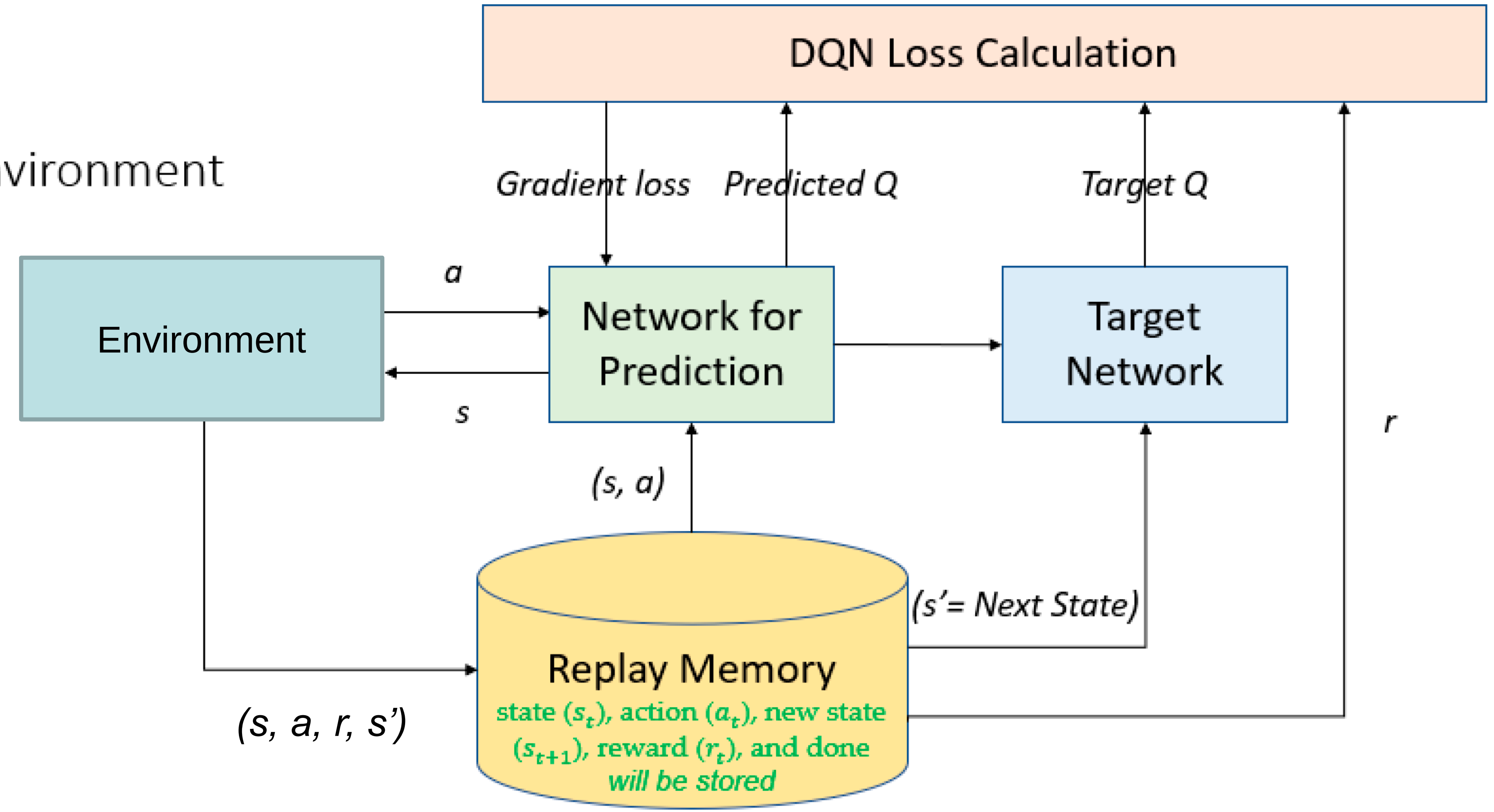
Take step to decrease error (in the direction of the negative gradient)

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

Overview of DQN

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

Environment



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

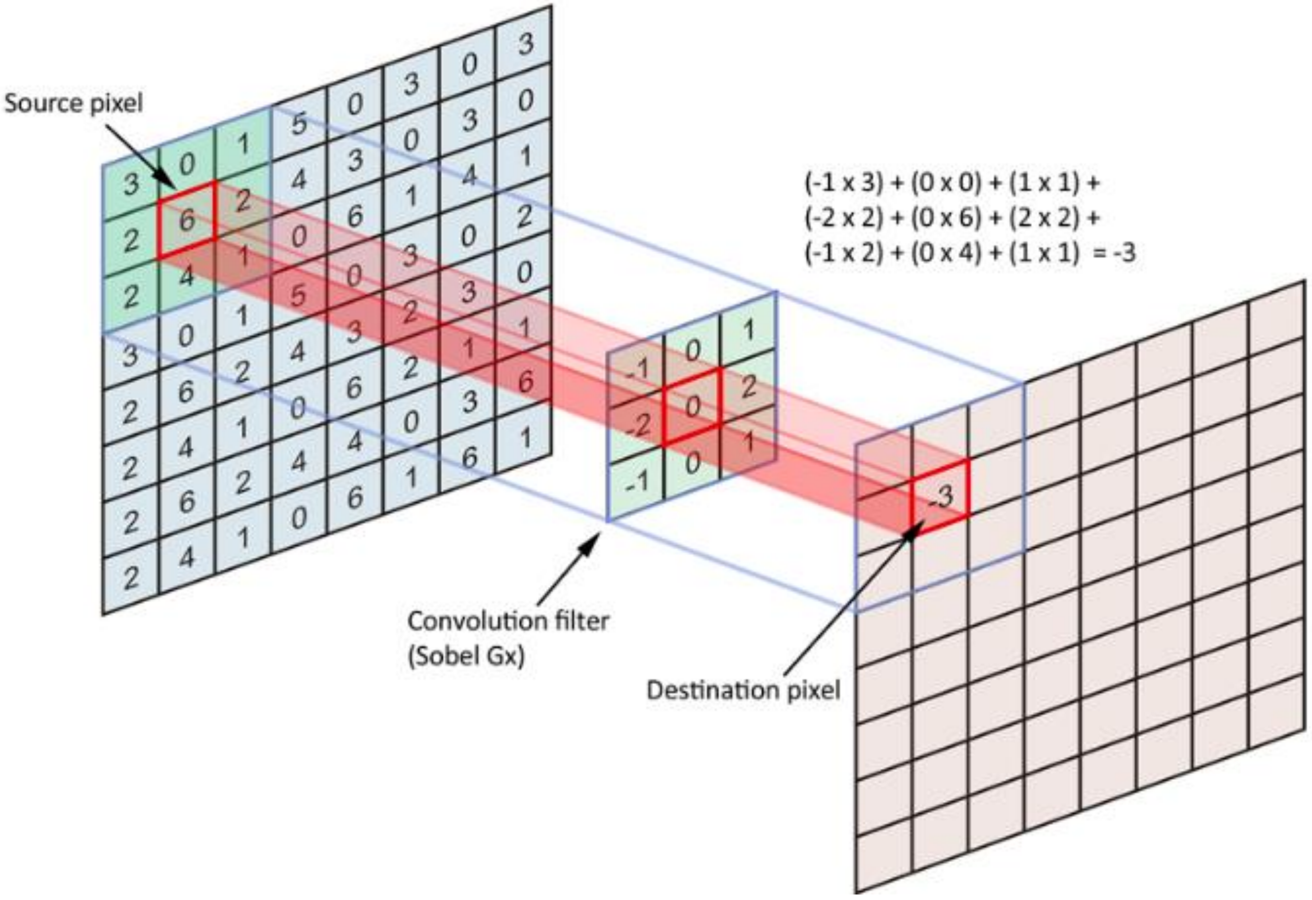
Catherine Shu @catherineshu / 6:20 PM MST • January 26, 2014

Comment

The Arcade Learning Environment

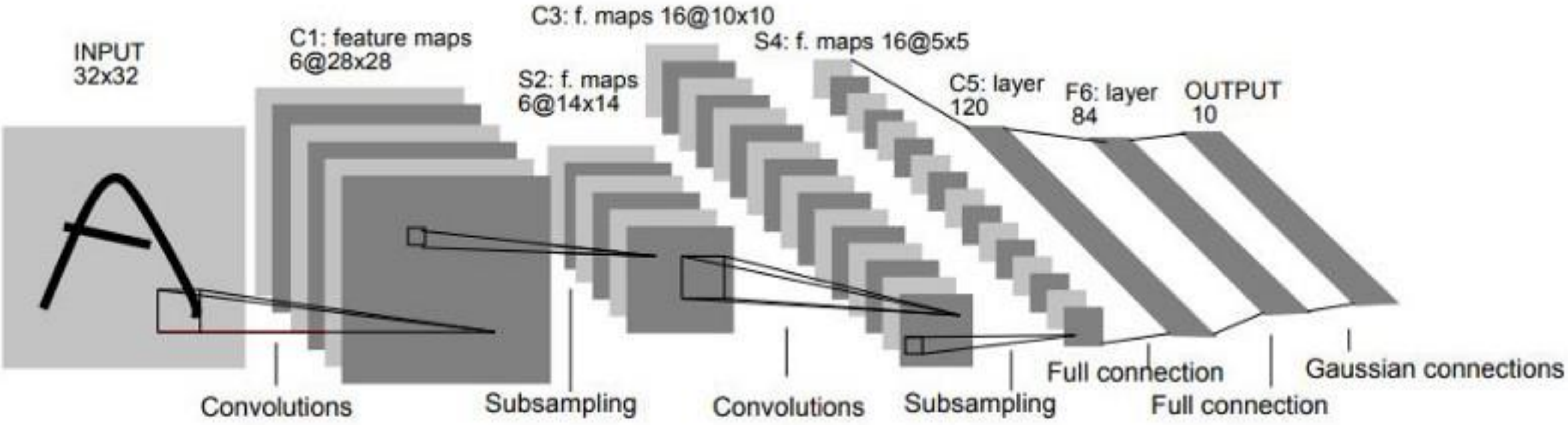
How do you learn from raw pixels?

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



How do you learn from raw pixels?

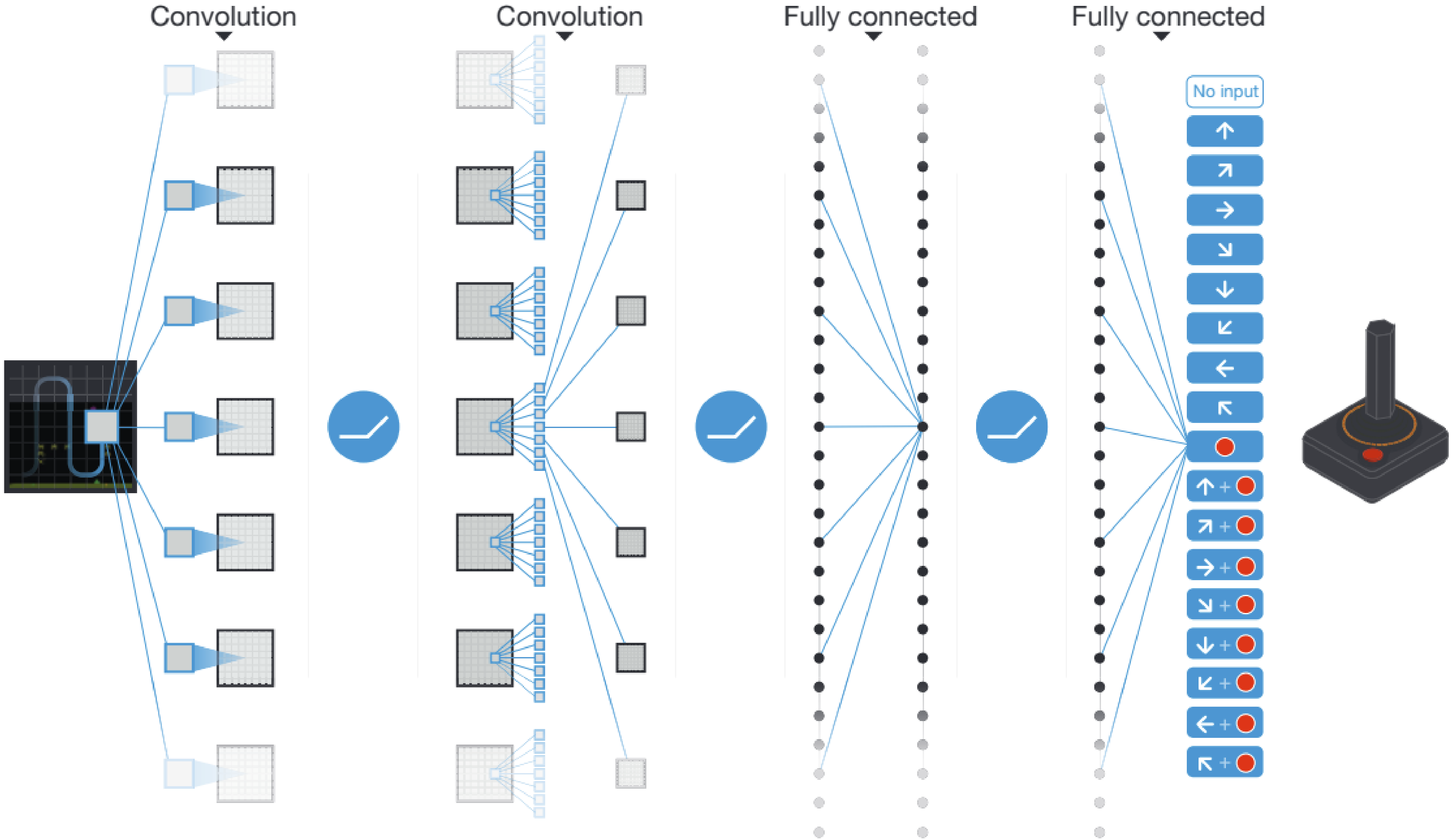
- Too many parameters to have a weight for each pixel.
- Use a convolutional filter
- Use several layers of multiple filters



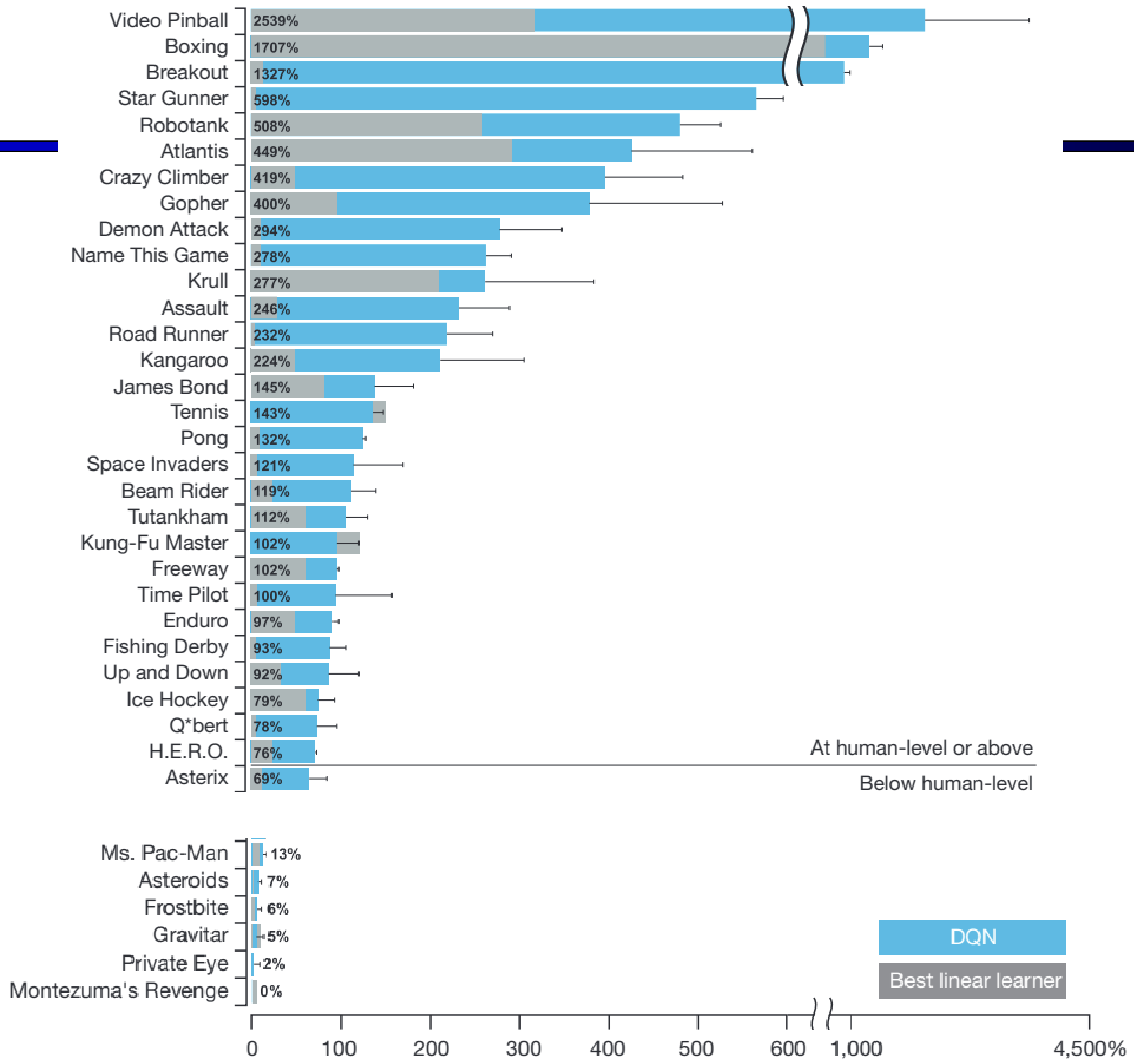
LeCun, Yann, et al. "Gradient-based learning applied to document recognition." 1998.

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%


0
We trained DQN with
novelty-based rewards.

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

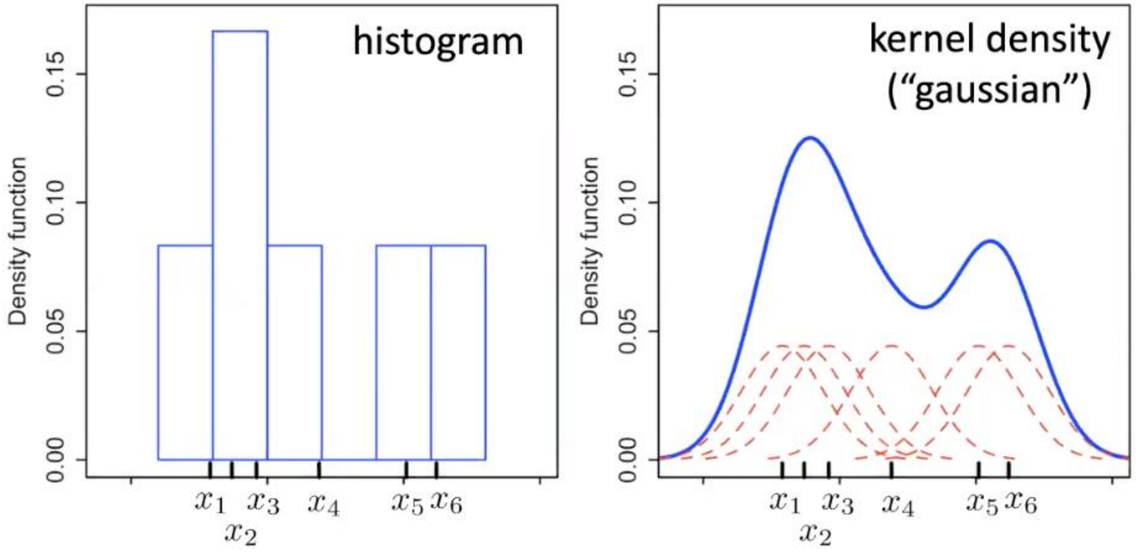
  - Note: this propagates the "bonus" back to states that lead to unknown states as well!

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

Generalizing Count-Based Exploration

- Normal counts of a state: $\frac{N_n(s)}{n}$

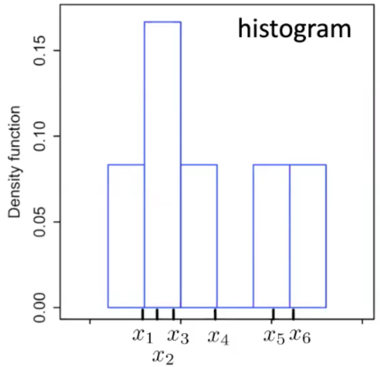
- Pseudo-Counts:
  - First assume access to a density model ρ that measures the probability of a state.



Generalizing Count-Based Exploration

- Normal counts of a state: $\frac{N_n(s)}{n}$

- Pseudo-Counts:
  - First assume access to a density model ρ that measures the probability of a state.
  - Define $\rho_n(s) = \rho(s|s_{1:n})$ as the probability of the (n+1)-th state being s given the first n states.
    - We could empirically estimate this as $\rho_n(s) = \frac{N_n(s)}{n}$



Generalizing Count-Based Exploration

- Pseudo-Counts:
  - First assume access to a density model ρ that measures the probability of a state.
  - Define $\rho_n(s) = \rho(s|s_{1:n})$ as the probability of the (n+1)-th state being s given the first n states.
    - We could empirically estimate this as $\rho_n(s) = \frac{N_n(s)}{n}$
  - Define $\rho'_n(s) = \rho(s|s_{1:n}, s)$ as the probability of s given we see state s again.
    - We could empirically estimate this as $\rho'_n(s) = \frac{N_n(s)+1}{n+1}$

Generalizing Count-Based Exploration

- Pseudo-Counts:
  - $\rho_n(s) = \frac{N_n(s)}{n}$
  - $\rho'_n(s) = \frac{N_n(s)+1}{n+1}$

We don't know N or n and don't want to explicitly count them.

But it turns out we can solve the linear system for what they would be given the density models!

$$\hat{N}_n(s) = \hat{n}\rho_n(s) = \frac{\rho_n(s)(1-\rho'_n(s))}{\rho'_n(s)-\rho_n(s)}$$

0, 0.1
0.3, 0.31

Generalizing Count-Based Exploration

- Pseudo-Counts:
  - $\rho_n(s) = \rho(s|s_{1:n})$ estimated probability density before seeing state s
  - $\rho'_n(s) = \rho(s|s_{1:n}, s) = \rho_{n+1}$ estimated probability density after updating density given new observation of s

$$\hat{N}_n(s) = \hat{n}\rho_n(s) = \frac{\rho_n(s)(1 - \rho'_n(s))}{\rho'_n(s) - \rho_n(s)}$$

  - Reward bonus is added to sparse true reward

$$R_n^+(x, a) := \beta(\hat{N}_n(x) + 0.01)^{-1/2}$$


0
We trained DQN with
novelty-based rewards.

DQN only works for discrete action spaces

- Next Time: How to deal with continuous action spaces