APPROXIMATE Q-LEARNING

Scott O'Hara
Metrowest Developers Machine Learning Group
06/10/2020

REFERENCES

CS188 - Introduction to Artificial Intelligence course at University of California, Berkeley:

- **http://ai.berkeley.edu/home.html**
- **http://gamescrafters.berkeley.edu/~cs188/sp20/**

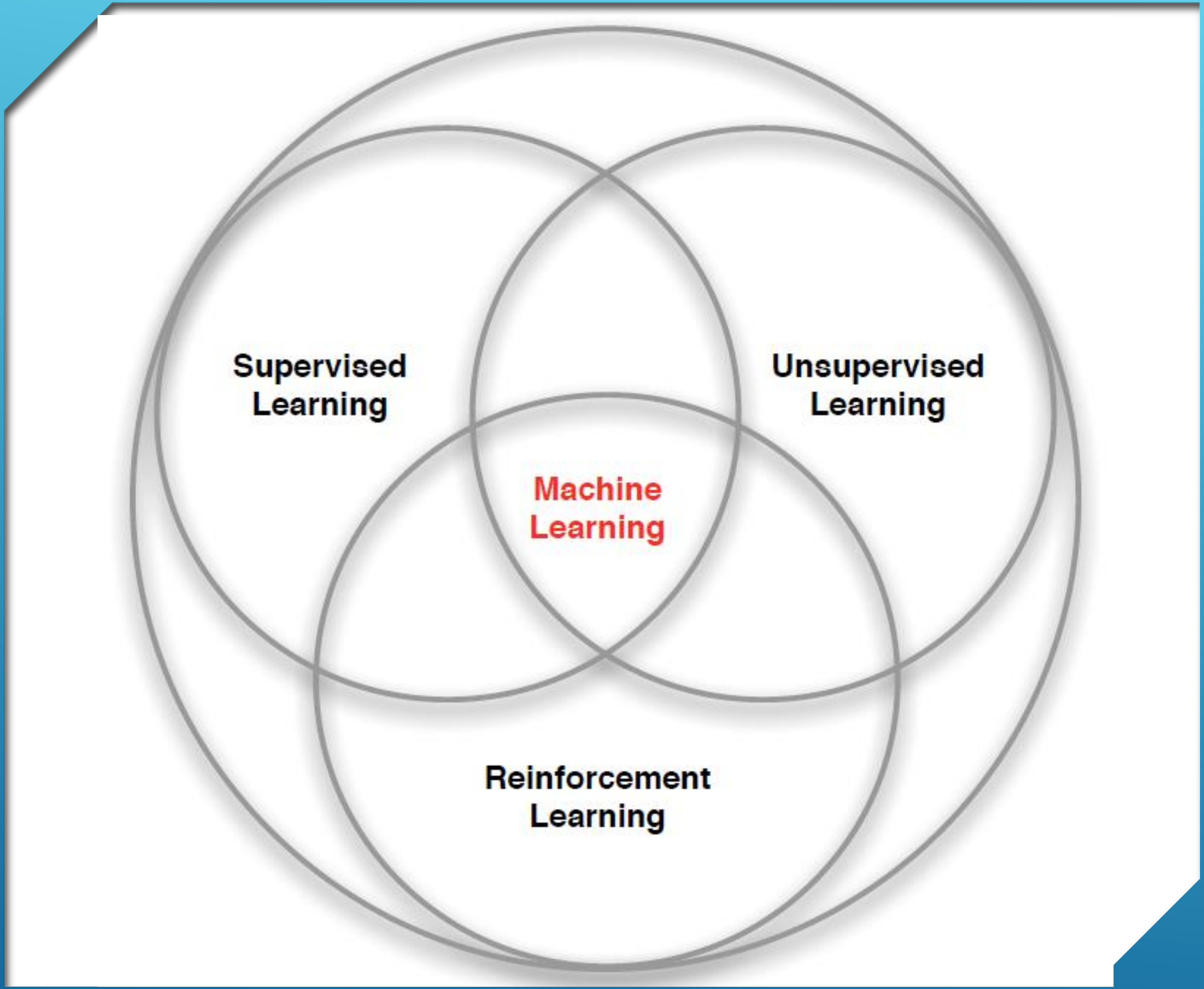
CS181 - Machine Learning course at Harvard University:

- ***Lectures and notes from multiple offerings of CS181: Spring 2009, 2011, 2014 and 2017.***

***The Hundred-Page Machine Learning Book*, A. Burkov, Publisher: Andriy Burkov, 2019. http://themlbook.com/ .**

***Reinforcement learning: an Introduction*, R. S. Sutton and A. G. Barto, Second edition. Cambridge, Massachusetts: The MIT Press, 2018.**

REINFORCEMENT LEARNING



3 BRANCHES OF MACHINE LEARNING

Credit: adapted from lecture slides David Silver, DeepMind, “Introduction to Reinforcement Learning

1. SUPERVISED LEARNING

- $\boldsymbol{Labeled\ Data} = \{(\boldsymbol{x_1}, \boldsymbol{y_1}), \ldots, (\boldsymbol{x_n}, \boldsymbol{y_n})\}$ with ***feature vector*** $\boldsymbol{x_i}$ and ***label*** $\boldsymbol{y_i}$.
- ***Train*** a ***model*** f on the labeled data.
- ***Predict*** label $y = f(x)$ for unlabeled data.
- Examples: linear regression, decision trees, SVMs, k-nearest neighbors.

Credit: adapted *The Hundred-Page Machine Learning Book, A. Burkov, 2019.*

2. UNSUPERVISED LEARNING

- $\boldsymbol{Unlabeled\ Data} = \{\boldsymbol{x_1}, , \ldots, \boldsymbol{x_n}\}$ of ***feature vectors*** $\boldsymbol{x_i}$.

- Create a ***model*** f on the unlabeled data.

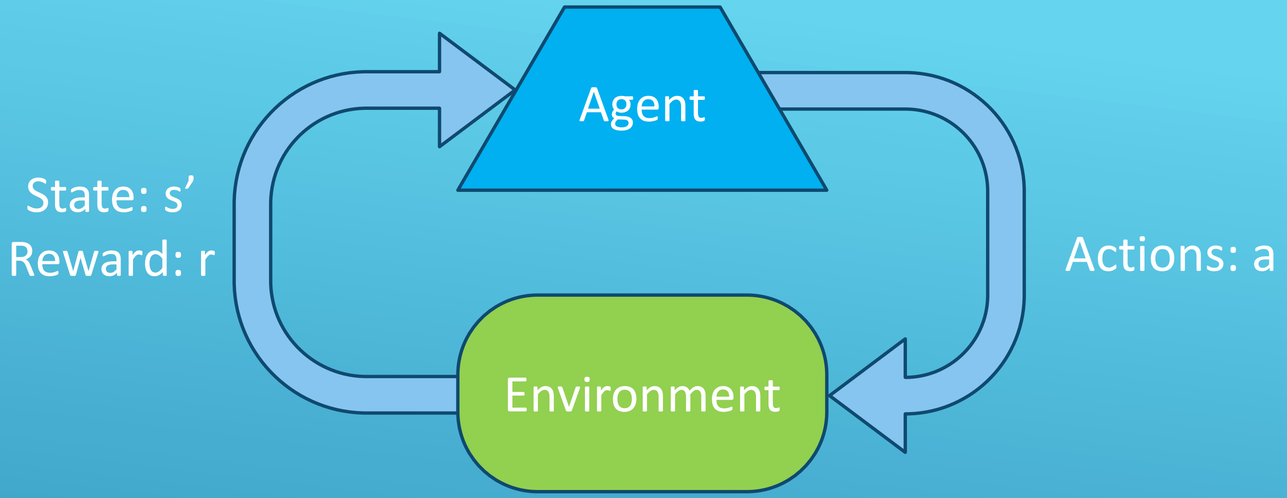
- ***Transform*** feature vector x into:
  - ***ID*** $y = f(x)$ ***(Clustering)***
  - ***Vector*** $\mathrm{x}' = f(x)$ ***(Dimensionality Reduction)***

- Clustering examples: K-means, HAC

- Dimensionality reduction examples: PCA, UMAP

Credit: adapted *The Hundred-Page Machine Learning Book, A. Burkov, 2019.*

3. REINFORCEMENT LEARNING



- Agent “lives” in an environment and can perceive the ***state*** of that environment as a ***vector of features***.
- Agent learns to act to ***maximize the expected average reward***
- Agent knows the current state *s*, takes an action ***a***, receives a reward ***r*** and observes the next state ***s’*.**

$$\boldsymbol{S_0, A_0, R_0, S_1, A_1, R_2, S_2, A_2, R_2, \dots, S_n, A_n, R_n, S_T}$$

Credit: adapted *The Hundred-Page Machine Learning Book, A. Burkov, 2019.*

MARKOV DECISION PROCESSES

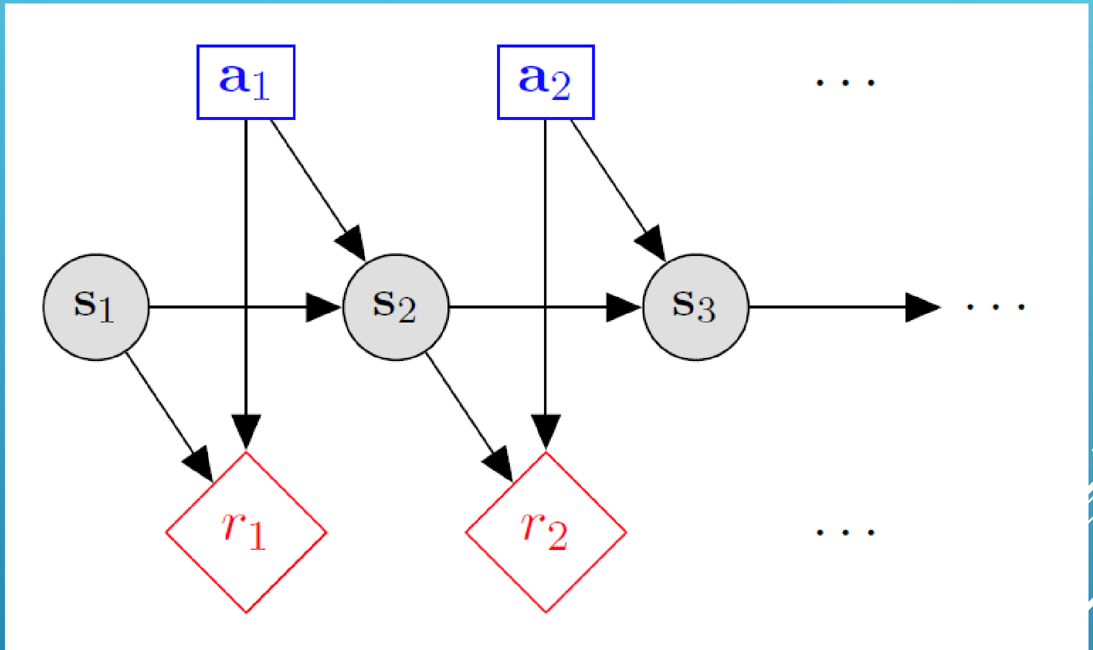
- **States:** $s_1, \ldots, s_n$
- **Actions:** $a_1, \ldots, a_m$
- **Reward model:**

  $$R(s, a, s') \in R$$

- **Transition model:**

  $$T(s, a, s') = P(s' | s, a)$$

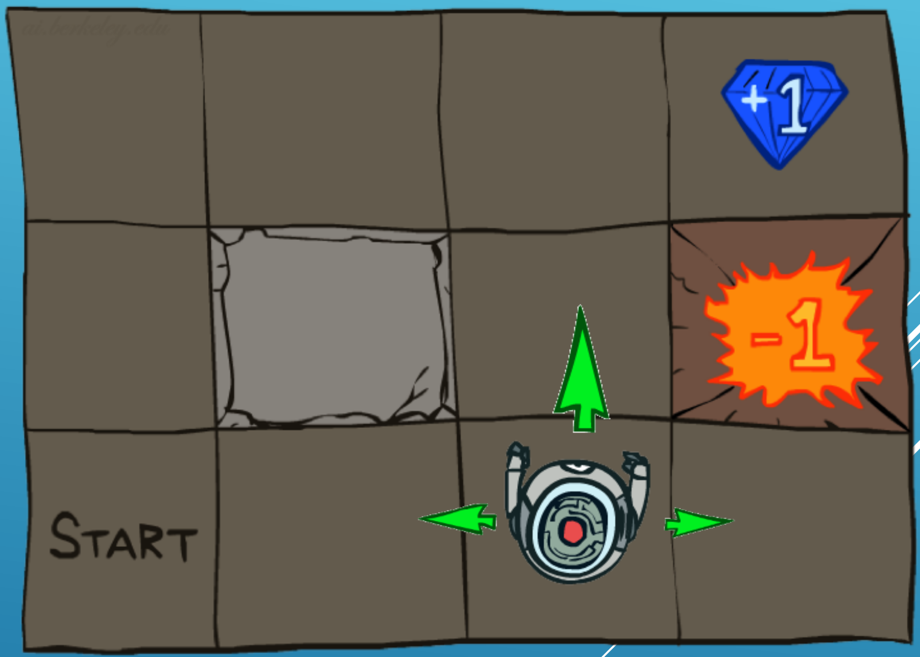
- **Discount factor:** $\gamma \in [0, 1]$



<https://en.wikipedia.org/w/index.php?title=Markov_decision_process&oldid=855934986> [accessed 11 September 2018]

EXAMPLE: GRIDWORLD

- A maze-like problem
  - The agent lives in a grid
  - Walls block the agent's path
- Noisy movement: actions do not always go as planned
  - 80% of the time, the action North takes the agent North (if there is no wall there)
  - 10% of the time, North takes the agent West; 10% East
  - If there is a wall in the direction the agent would have been taken, the agent stays put
- The agent receives rewards each time step
  - Small "living" reward each step (usually negative)
  - Big rewards come at the end (good or bad)
- Goal: maximize sum of rewards



EXAMPLE: GRIDWORLD

- **States: <x, y>** locations
- **Actions:** move **north, south, east** or **west.**
- **Reward model:**

  +1 if robot moves to <3, 2>

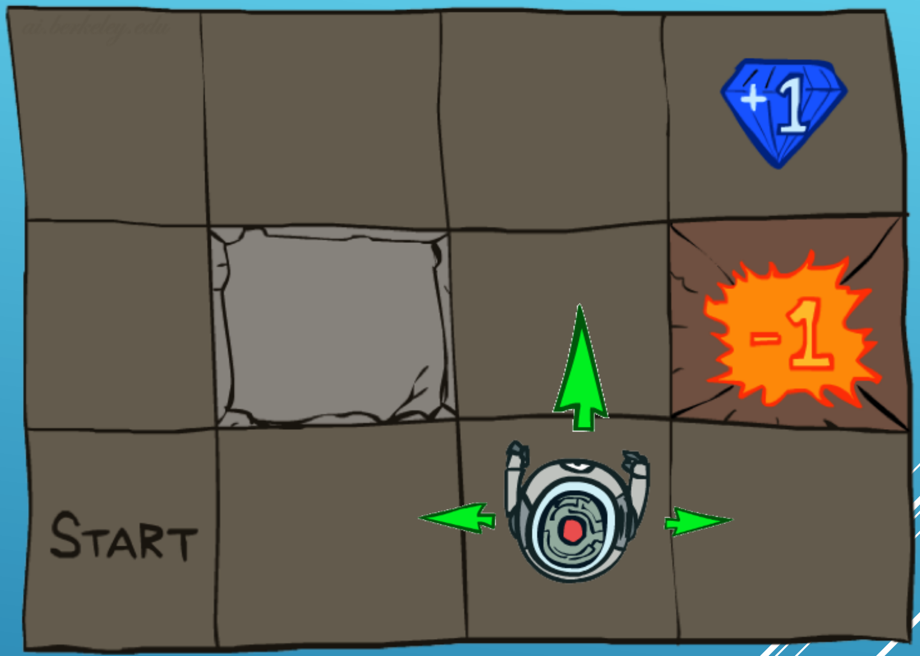
  -1 if robot moves <3, 1>

  otherwise, get "living reward".
- **Transition model:**

  n – probability of unintended (noisy) action.

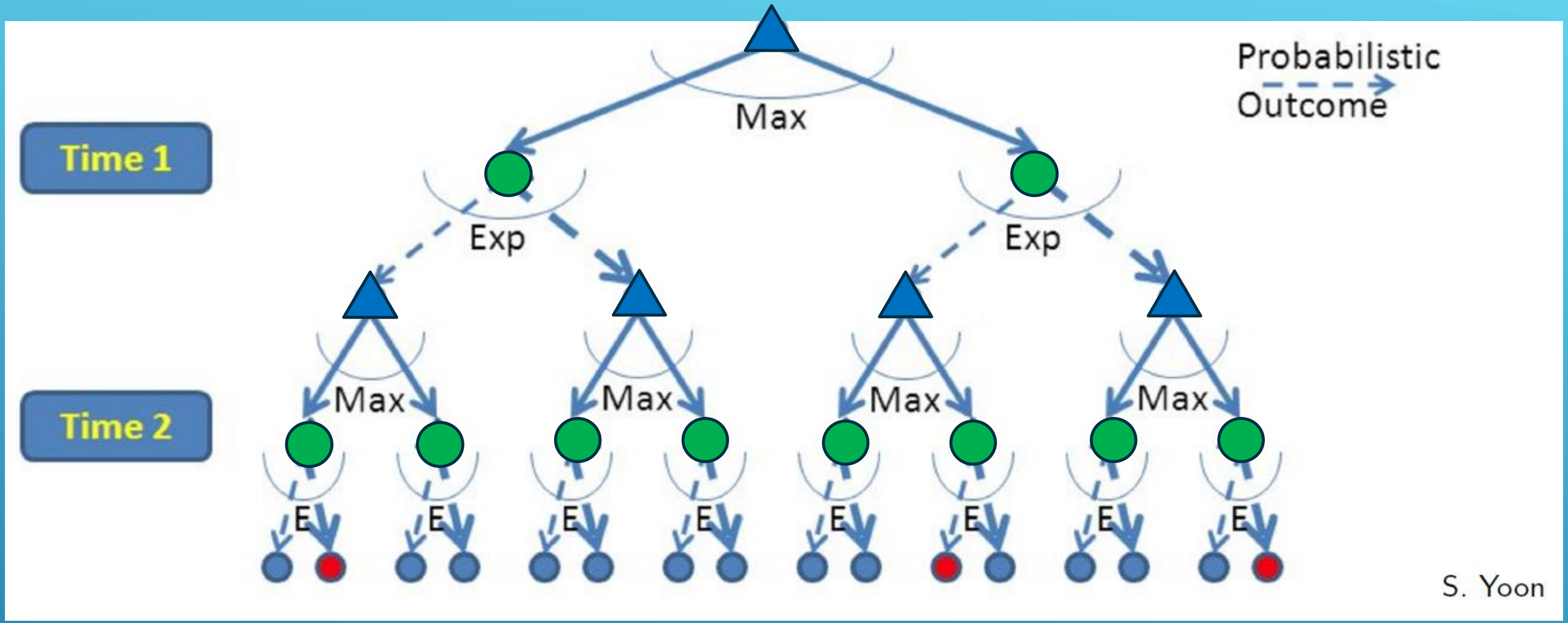
  1-n probability of intended action.

  stay put if you move into a wall.



TABULAR Q-LEARNING

RL IS LIKE A GAME AGAINST NATURE



- Reinforcement learning is like a game-playing algorithm.
- Nodes where you move are called **states**: S (▲)
- Nodes where nature "moves" are called **Q-states**: <S,A> (●)

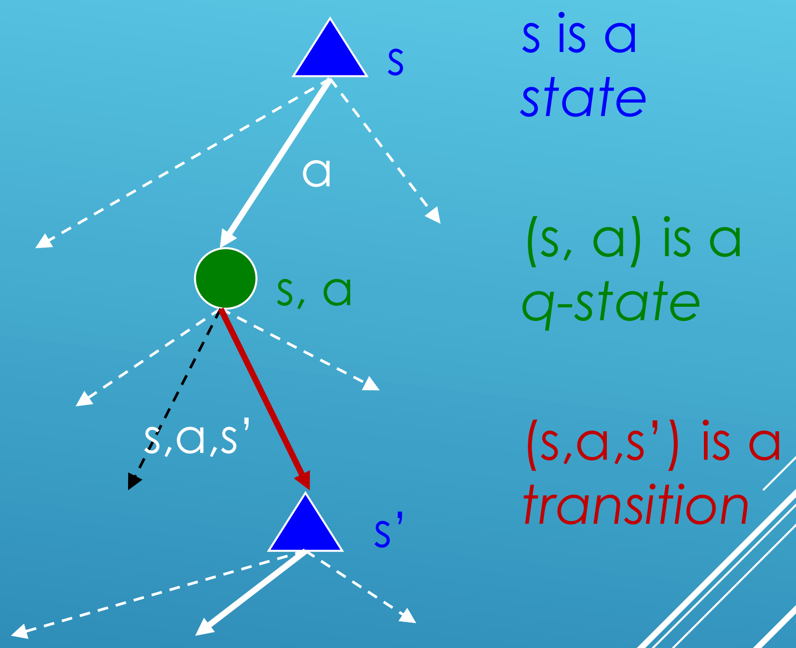
Credit: CS188 – Intro to AI, UC Berkeley, ai.berkely.edu

QUANTITIES TO OPTIMIZE

- The **state-value function:**
  V(s) = the expected utility of starting in state *s* and acting optimally afterwards.

- The **action-value function**:
  Q(s,a) = the expected utility of starting in state *s*, taking action *a,* and acting optimally afterwards.

- The **policy**:
  π(s) = defines the action to take in every state s.



Credit: CS188 – Intro to AI, UC Berkeley, ai.berkeley.edu

THE BELLMAN OPTIMALITY EQUATIONS

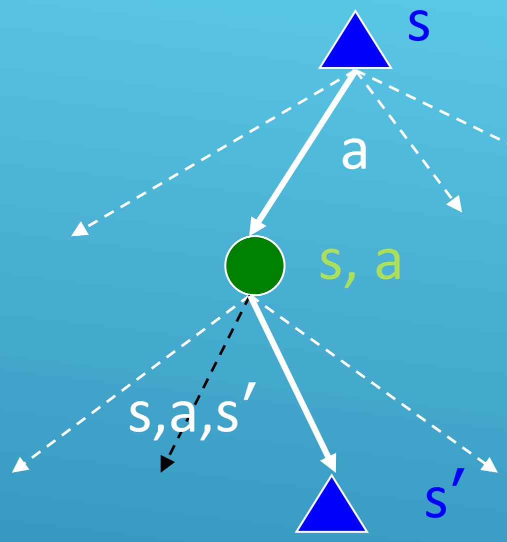
- The Bellman Optimality Equations define a relationship that, when satisfied, guarantees that the ***state-value function*** $\boldsymbol{V^*(s)}$ and the ***state-action function*** $\boldsymbol{Q^*(s, a)}$ are optimal for every state and action.
- This in turn guarantees that the policy π is optimal, which is designated π^*.

Credit: CS188 – Intro to AI, UC Berkeley, ai.berkeley.edu
Credit: CS181 – Machine Learning, D. Parkes, Harvard Univ, Spring 2017

THE BELLMAN OPTIMALITY EQUATIONS

State-Value Equation:

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



Action-Value Equation:

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

Simplifying Assumption: rewards are fixed for (s, a, s')

Credit: CS188 – Intro to AI, UC Berkeley, ai.berkeley.edu

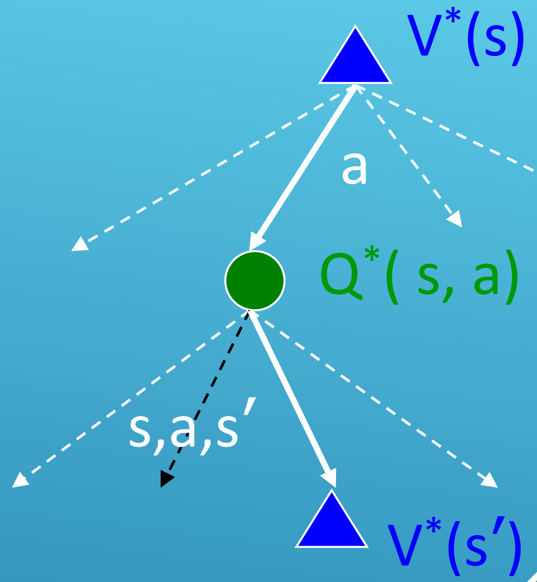
Credit: CS181 – Machine Learning, D. Parkes, Harvard Univ, Spring 2017

MUTUAL RECURSION

State-Value Equation:

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

$$\boxed{V^*(s)} = \boxed{\max_a} \boxed{Q^*(s,a)}$$



Action-Value Equation:

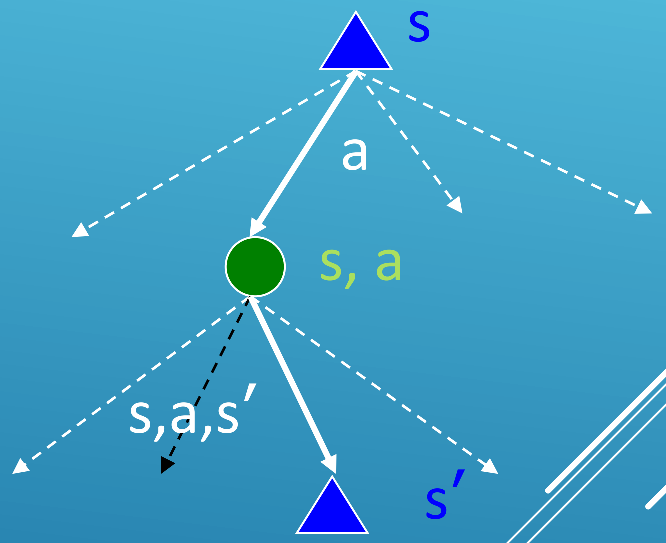
$$\boxed{Q^*(s,a)} = \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \underbrace{\boxed{\max_{a'} Q^*(s',a')}}\right]$$

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

THE OPTIMAL STATE-VALUE EQUATION V*

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

- Focusing on different Bellman Equations gives different algorithms
- The V* equation gives rise to these dynamic programming algorithms previously discussed:
  - Value Iteration
  - Policy Iteration



Credit: CS188 – Intro to AI, UC Berkeley, ai.berkeley.edu

Credit: CS181 – Machine Learning, D. Parkes, Harvard Univ, Spring 2017

THE OPTIMAL VALUE UTILITY EQUATION Q*

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

The Q* equation gives rise to the Q-Learning algorithm.

Credit: CS188 – Intro to AI, UC Berkeley, ai.berkeley.edu
Credit: CS181 – Machine Learning, D. Parkes, Harvard Univ, Spring 2017

FROM EQUATION TO UPDATE RULE

- What to do about *T(s,a,s')* and *R(s,a,s')*, since we don't have these functions?

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

- Use sampling to learn Q(s,a) values as you go
  - Receive a sample transition: ***(s,a,r,s')***
  - Consider your old estimate: **Q**(s,a)
  - Consider your new sample estimate: $r + \gamma \max_{a'} Q_k(s',a')$
  - Incorporate the new estimate into a running average based on the learning rate α:

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

Q-LEARNING UPDATE RULE (2)

- On transitioning from state *s* to state *s'* on action *a*, and receiving reward *r*, update:

$$\boldsymbol{Q_{k+1}(s,a) \leftarrow Q_k(s,a) + \alpha\Big[\underbrace{r + \gamma \max_{a'} Q_k(s',a')}_{\text{target}} - \underbrace{Q_k(s,a)}_{\text{current}}\Big]}$$

- α is the **learning rate**
  - A large α results in quicker learning but may not converge.
  - α is often decreased as learning goes on.

- γ is the **discount rate** i.e., discounts future rewards.

CHOOSING AN ACTION: EXPLORATION VS EXPLOITATION

- How should an agent choose an action? An obvious answer is simply to follow the current policy. However, this is often not the best way to improve your model.
- **Exploit:** use your current model to maximize the expected utility now.
- **Explore:** choose an action that will help you improve your model.

Ε-GREEDY METHOD

- With probability 1 - ϵ:

  $A_t = argmax\ Q_t(s, a)$, select action with maximum value.

- With probability ϵ:

  $A_t =$ select with equal probability an action at state s from all possible actions.

TABULAR Q-LEARNING ALGORITHM

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

Credit: *Reinforcement Learning: an Introduction, R. Sutton and A. Barto, 2018.*

TRANSITION/REWARD PARAMETER TABLES

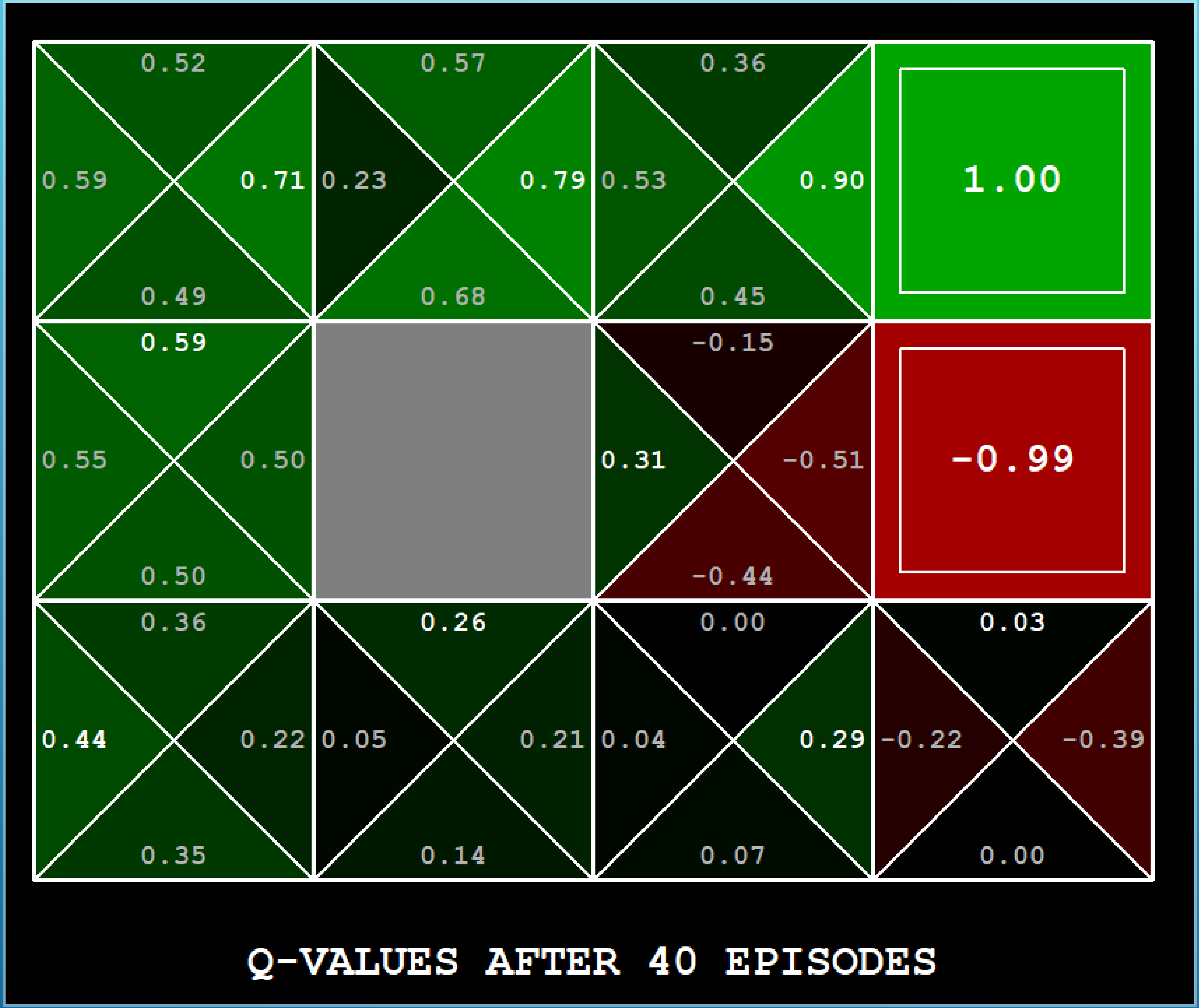
For every state s, there is a transition table **T** and a reward table **R**:

State s'

Action a

$\hat{T}(s,a0,s0)$ $\hat{R}(s,a0,s0)$	$\hat{T}(s,a0,s1)$ $\hat{R}(s,a0,s1)$	$\hat{T}(s,a0,s2)$ $\hat{R}(s,a0,s2)$	$\hat{T}(s,a0,s3)$ $\hat{R}(s,a0,s3)$
$\hat{T}(s,a1,s0)$ $\hat{R}(s,a1,s0)$	$\hat{T}(s,a1,s1)$ $\hat{R}(s,a1,s1)$	$\hat{T}(s,a1,s2)$ $\hat{R}(s,a1,s2)$	$\hat{T}(s,a1,s3)$ $\hat{R}(s,a1,s3)$
$\hat{T}(s,a2,s0)$ $\hat{R}(s,a2,s0)$	$\hat{T}(s,a2,s1)$ $\hat{R}(s,a2,s1)$	$\hat{T}(s,a2,s2)$ $\hat{R}(s,a2,s2)$	$\hat{T}(s,a2,s3)$ $\hat{R}(s,a2,s3)$

Q-LEARNING EXAMPLE: GRIDWORLD

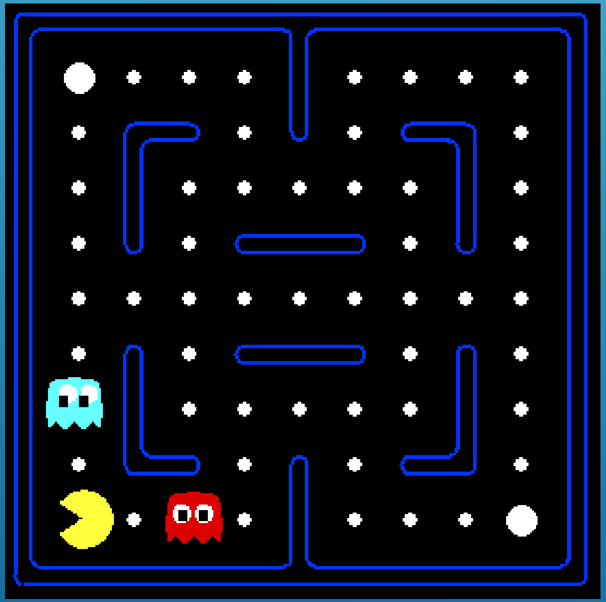


APPROXIMATE REINFORCEMENT LEARNING

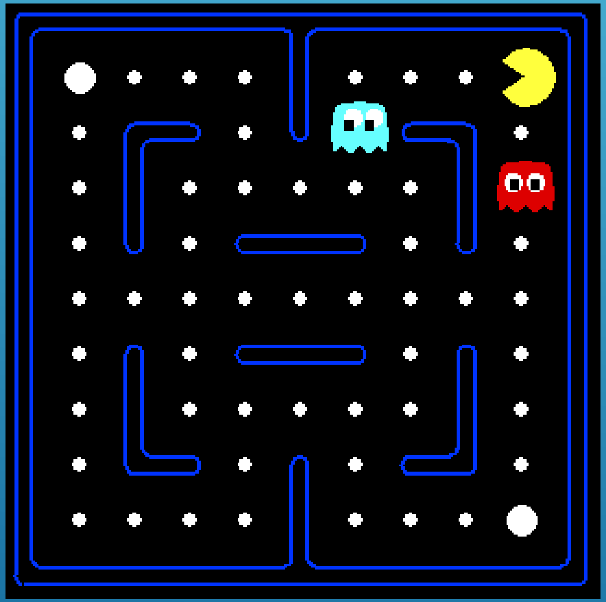
- Problems with tabular reinforcement learning:
  - Large state spaces need large amounts of memory.
  - Large state spaces require a long time to fill with accurate values.
- Key question: How can experience with a limited subset of the state space be usefully generalized to produce a good approximation over a much larger subset?
- Generalization can be accomplished with function approximation, which is an instance of supervised learning.
- In theory, any of the supervised learning techniques can be used as a function approximator, though some approaches are better than others.

EXAMPLE: PACMAN

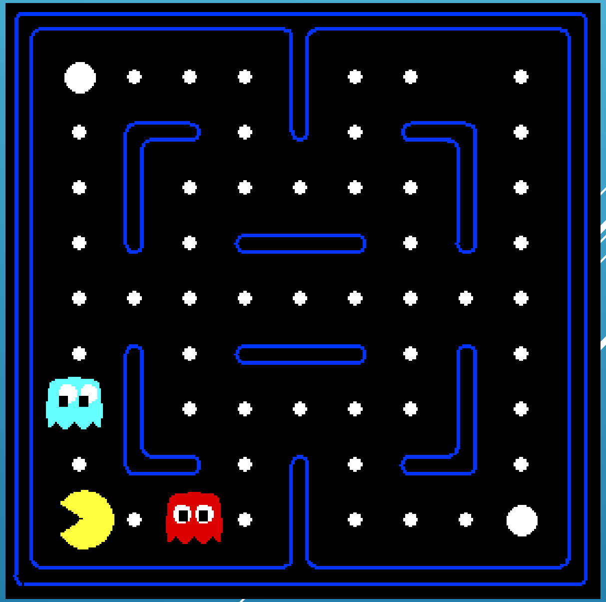
Let's say we discover through experience that this state is bad:



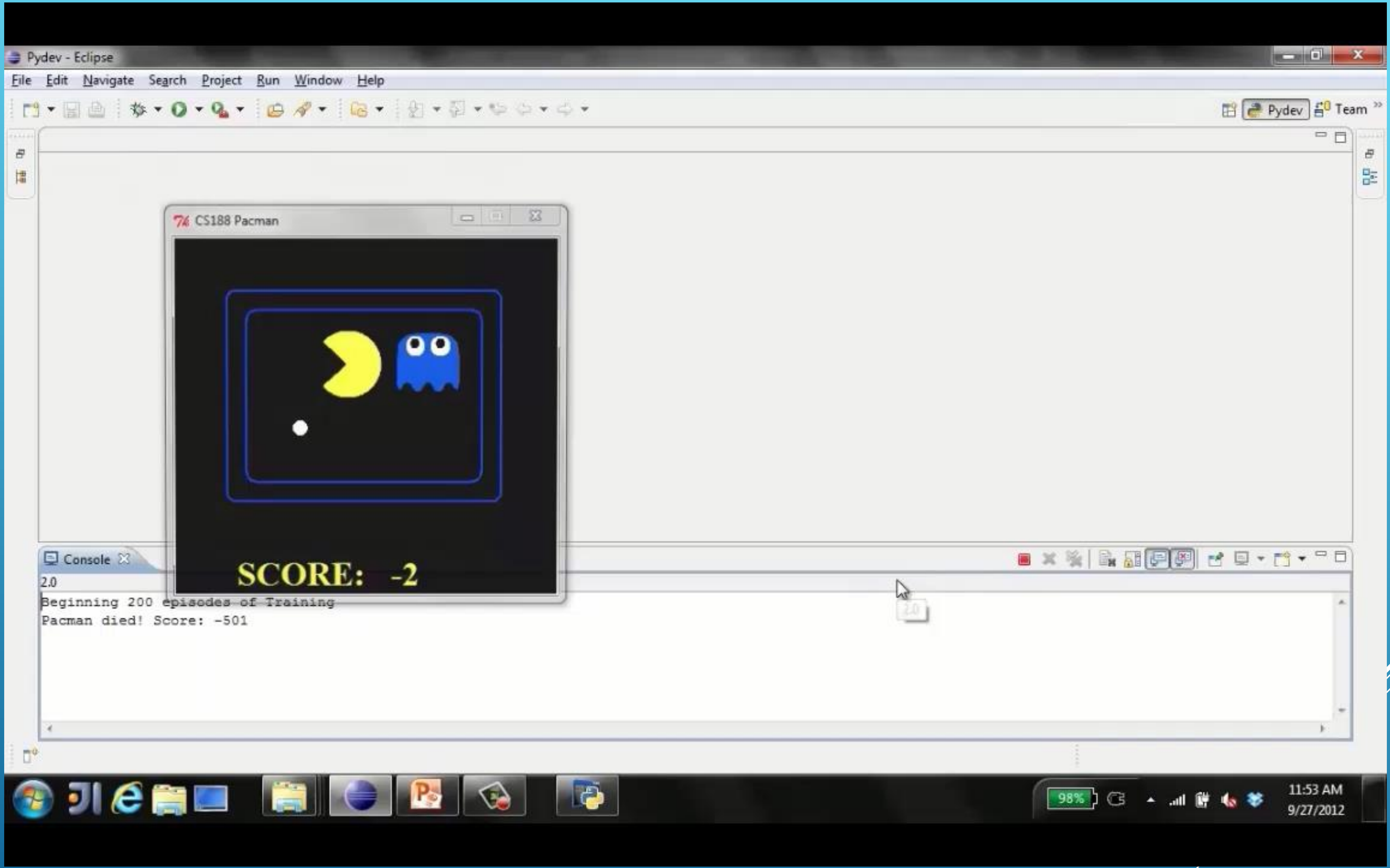
In naïve q-learning, we know nothing about this state:



Or even this one!

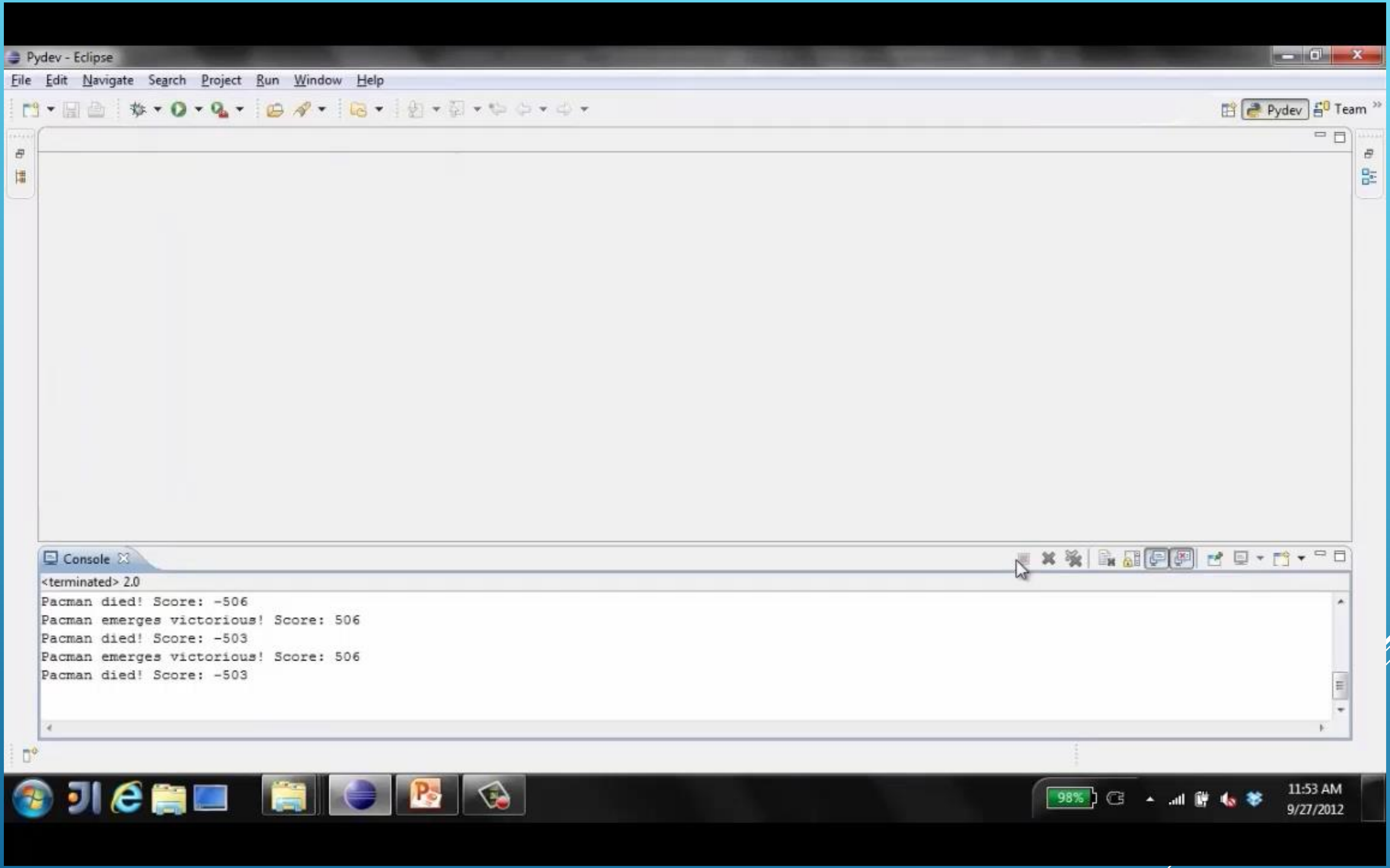


Credit: CS188 – Intro to AI, UC Berkeley, ai.berkeley.edu



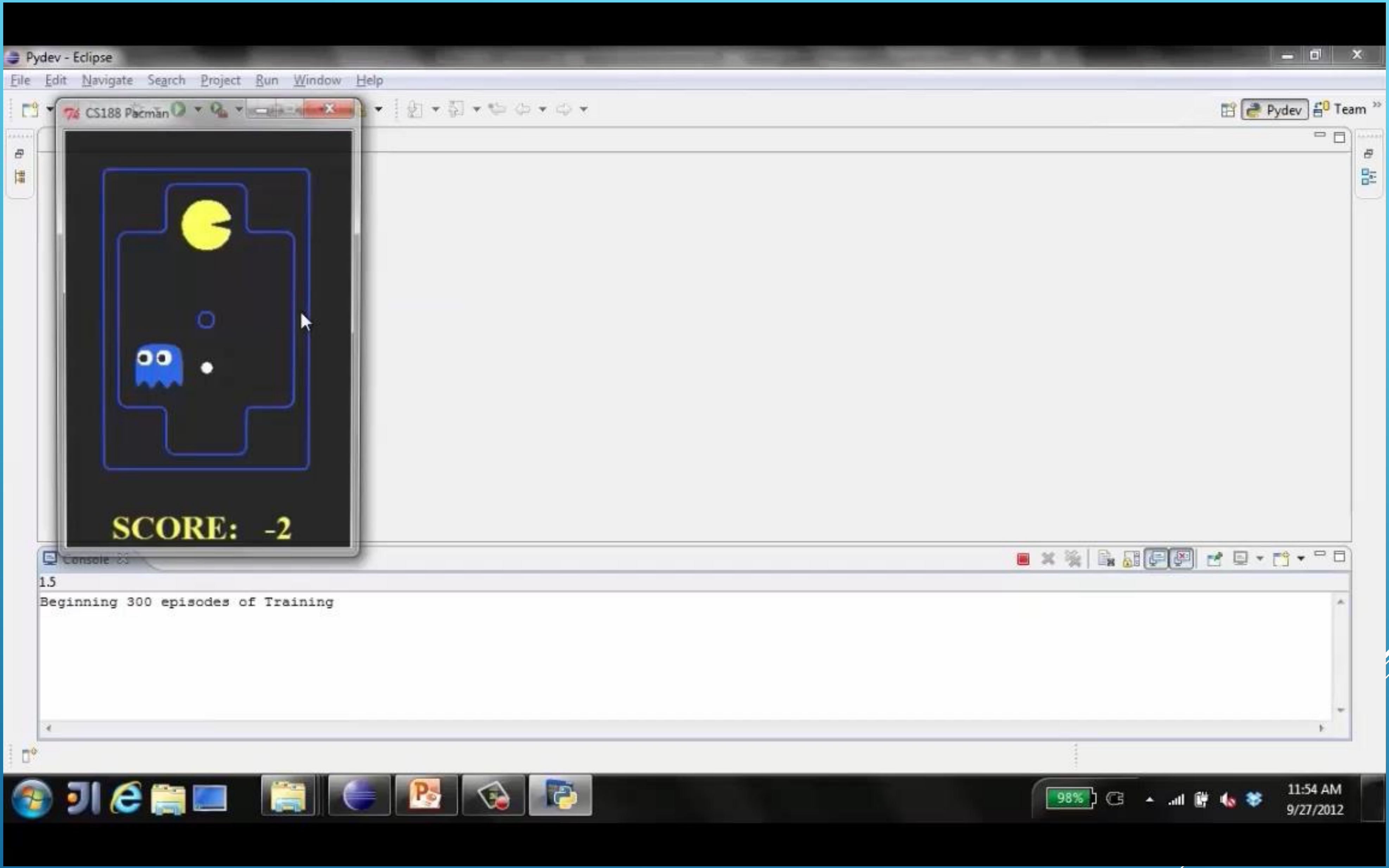
TINY PACMAN DEMO 1

Credit: CS188 – Intro to AI, UC Berkeley, ai.berkeley.edu



TINY PACMAN DEMO 2

Credit: CS188 – Intro to AI, UC Berkeley, ai.berkeley.edu



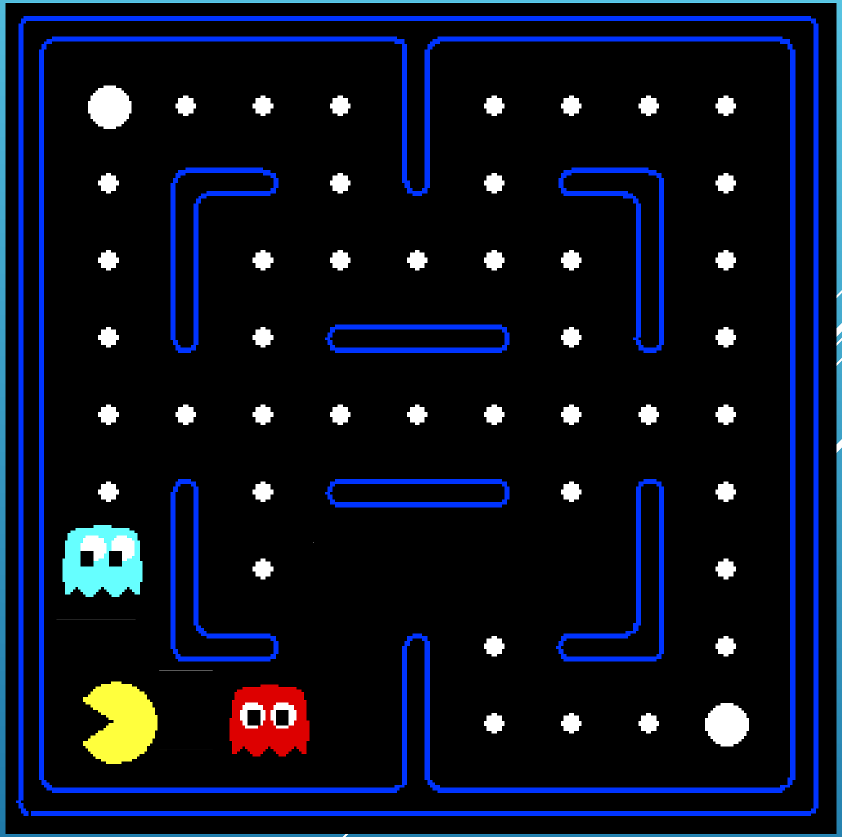
TINY PACMAN DEMO 3

Credit: CS188 – Intro to AI, UC Berkeley, ai.berkely.edu

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)
  - Is it the exact configuration on this slide?
  - …… etc.
- Can also describe a q-state (s, a) with features (e.g. action moves closer to food)



Credit: CS188 – Intro to AI, UC Berkeley, ai.berkeley.edu

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!

Credit: CS188 – Intro to AI, UC Berkeley, ai.berkeley.edu

Q-LEARNING WITH LINEAR Q-FUNCTIONS

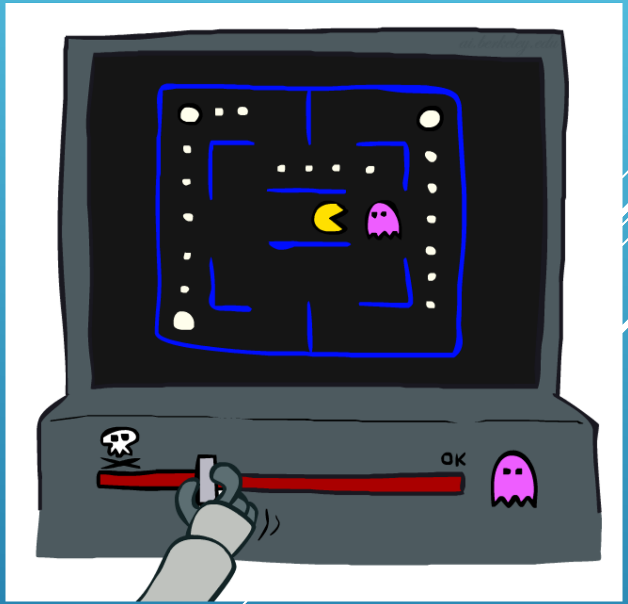
$$\boldsymbol{Q(s,a,\overline{w}) = w_1 x_1(s,a) + w_2 x_2(s,a) + \ldots + w_n x_n(s,a)}$$

Exact Q's:

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

Approximate Q's:

- $\boldsymbol{Q(s,a,\overline{w}) = \overline{w}^T \overline{x}}$
- $\boldsymbol{\nabla Q(s,a,\overline{w}) = \overline{x}}$ for linear functions
- $\boldsymbol{w_i} \leftarrow \boldsymbol{w_i} + \alpha[\text{difference}]\frac{\partial\, \mathrm{x}_i(s,a)}{\partial w_i}$
- $\boldsymbol{w_i} \leftarrow \boldsymbol{w_i} + \alpha[\text{difference}]\mathrm{x}_i$

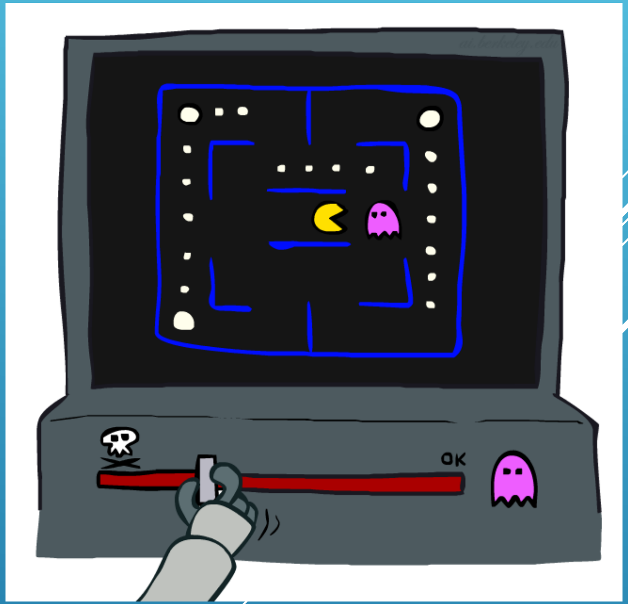


Credit: CS188 – Intro to AI, UC Berkeley, ai.berkeley.edu

INTUITIVE INTERPRETATION

$$\boldsymbol{w_i} \leftarrow \boldsymbol{w_i} + \alpha[\text{difference}]\frac{\partial\, \mathrm{x}_i(s,a)}{\partial w_i}$$

- Adjust weights of active features.
- If something good happens, increase the features that were on i.e., prefer similar states more that have that state's features.
- If something unexpectedly bad happens, blame the features that were on i.e., prefer similar states less that have that state's features.



Credit: CS188 – Intro to AI, UC Berkeley, ai.berkeley.edu

Q-LEARNING UPDATE RULES: TABULAR VS. LINEAR ACTION-VALUE FUNCTION

Tabular Update Rule:

$$Q_{k+1}(s,a) \leftarrow Q_k(s,a) + \alpha \left[\overbrace{r + \gamma \max_{a'} Q_k(s',a')}^{\text{target}} - \overbrace{Q_k(s,a)}^{\text{current}} \right]$$

Linear Action-Value Update Rule:

$$\bar{w} \leftarrow \bar{w} + \alpha \left[\overbrace{R + \gamma \max_{a'} \hat{q}(S',A',\bar{w})}^{\text{target}} - \overbrace{\hat{q}(S,A,\bar{w})}^{\text{current}} \right] \nabla \hat{q}(S,A,\bar{w})$$

APPROXIMATE Q-LEARNING ALGORITHM

Q-Learning

Episodic ~~Sarsa~~ with function approximation

Episodic Semi-gradient Sarsa for Estimating $\hat{q} \approx q_*$

Input: a differentiable action-value function parameterization $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$
Algorithm parameters: step size $\alpha > 0$, small $\varepsilon > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

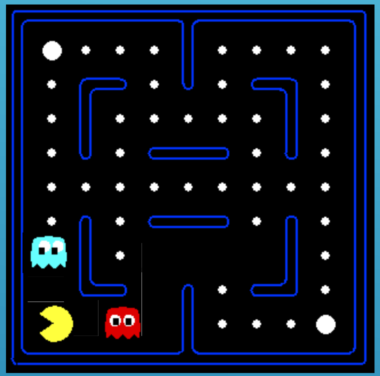
Loop for each episode:
  $S, A \leftarrow$ initial state and action of episode (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    If S' is terminal:
      $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$
      Go to next episode
    Choose A' as a function of $\hat{q}(S', \cdot, \mathbf{w})$ (e.g., ε-greedy)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \cancel{\gamma\hat{q}(S', A', \mathbf{w})} - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$
    $S \leftarrow S'$
    $A \leftarrow A'$

$\boldsymbol{\gamma \max_{a} \hat{q}\,(S', a, w)}$

EXAMPLE: Q-PACMAN

$$Q(s,a) = 4.0\,\mathrm{x}_{DOT}(s,a) - 1.0\,\mathrm{x}_{GST}(s,a)$$

s

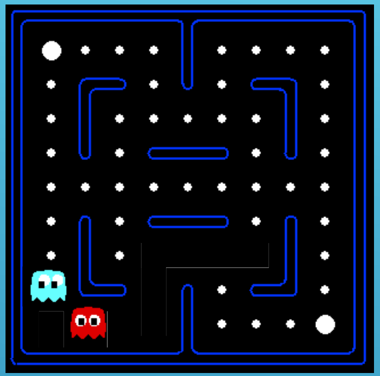


$\mathrm{x}_{DOT}(s, NORTH) = 0.5$

$\mathrm{x}_{GST}(s, NORTH) = 1.0$

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

s'



$Q(s, NORTH) = +1$

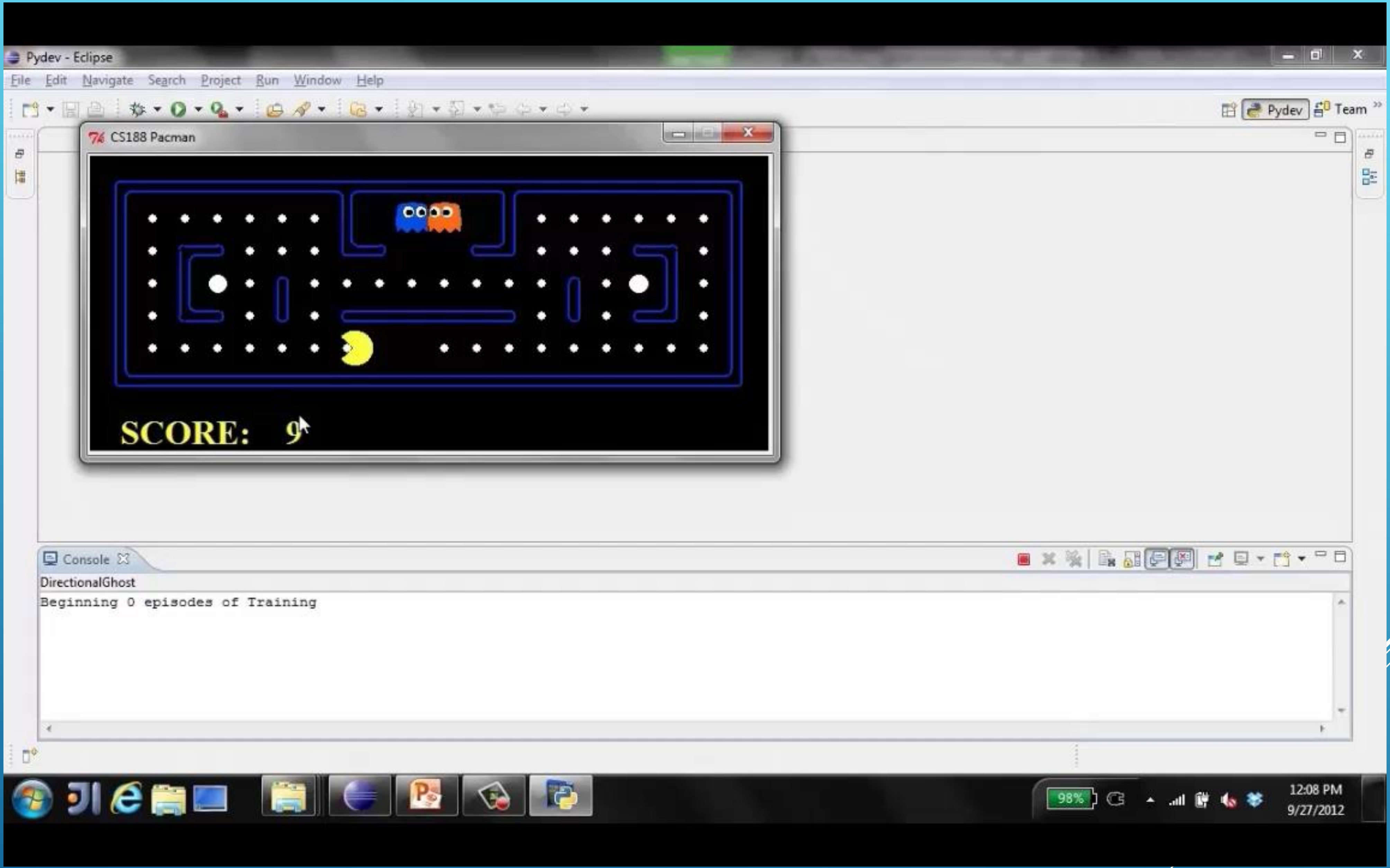
$r + \gamma \max_{a'} Q(s', a') = -500 - 1 = -501$

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

difference $= -501$

$\mathrm{w}_{DOT} \leftarrow 4.0 + \alpha[-501]0.5$
$\mathrm{w}_{GST} \leftarrow -1.0 + \alpha[-501]1.0$

$$Q(s,a) = 3.0\,\mathrm{f}_{DOT}(s,a) - 3.0\,\mathrm{x}_{GST}(s,a)$$



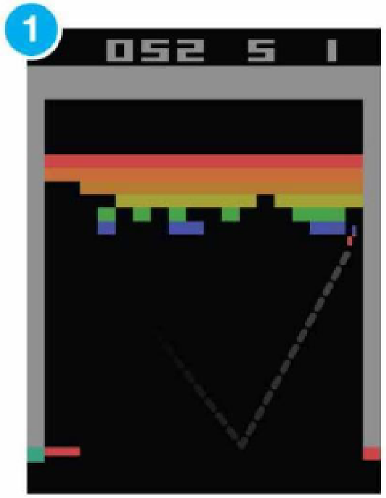
APPROXIMATE Q-LEARNING DEMO -- PACMAN

Credit: CS188 – Intro to AI, UC Berkeley, ai.berkeley.edu

DEEP Q-LEARNING: APPROXIMATE Q-LEARNING WITH DEEP NEURAL NETWORKS

Q-Learning: Representation Matters

- In practice, Tabular Q-Learning is impractical
  - Very limited states/actions
  - Cannot generalize to unobserved states

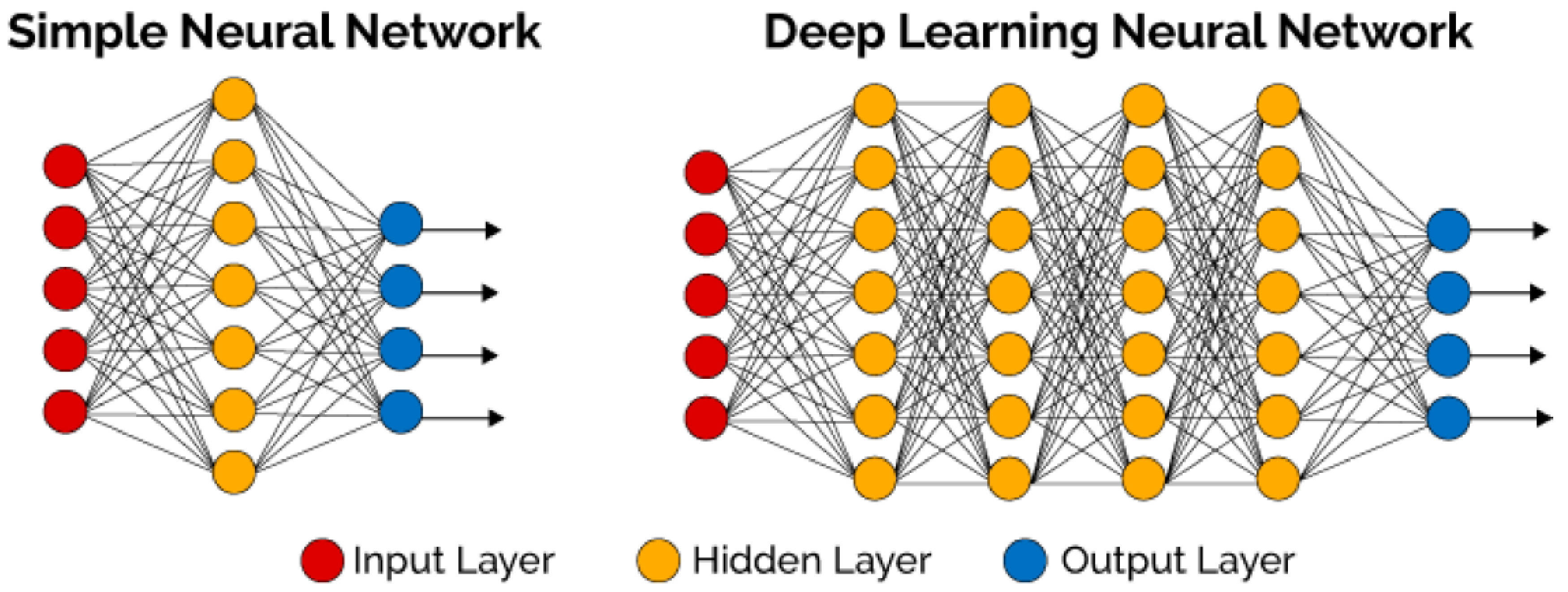


- Think about the **Breakout** game
  - State: screen pixels
    - Image size: 84×84 (resized)
    - Consecutive 4 images
    - Grayscale with 256 gray levels

$256^{84\times84\times4}$ rows in the Q-table!

$= 10^{69,970} >> 10^{82}$ atoms in the universe

Combing Neurons in Hidden Layers:
The "Emergent" Power to Approximate

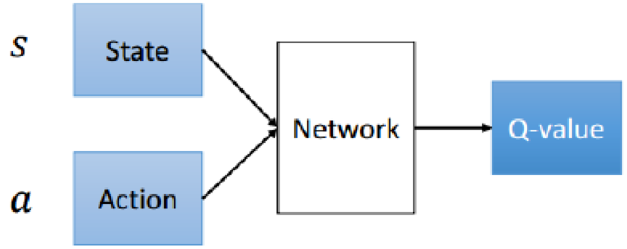


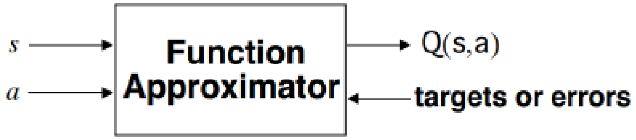
Universality: For any arbitrary function f(x), there exists a neural network that closely approximate it for any input x

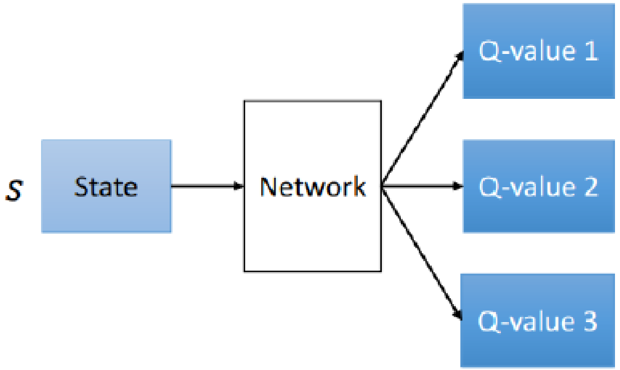
DQN: Deep Q-Learning

Use a neural network to approximate the Q-function:

$$Q(s, a; \boldsymbol{\theta}) \approx Q^*(s, a)$$







EXTRA SLIDES

EDITING^

Q-LEARNING UPDATE RULE: AN ALTERNATE INTERPRETATION

$$\boldsymbol{Q_{k+1}(s,a) \leftarrow Q_k(s,a) + \alpha\Big[\underbrace{r + \gamma \max_{a'} Q_k(s',a')}_{\text{target}} - \underbrace{Q_k(s,a)}_{\text{current}}\Big]}$$

$$\boldsymbol{Q_{k+1}(s,a) \leftarrow \underbrace{(1-\alpha)Q_k(s,a)}_{\text{old}} + \alpha\Big[\underbrace{r + \gamma \max_{a'} Q_k(s',a')}_{\text{new}}\Big]}$$