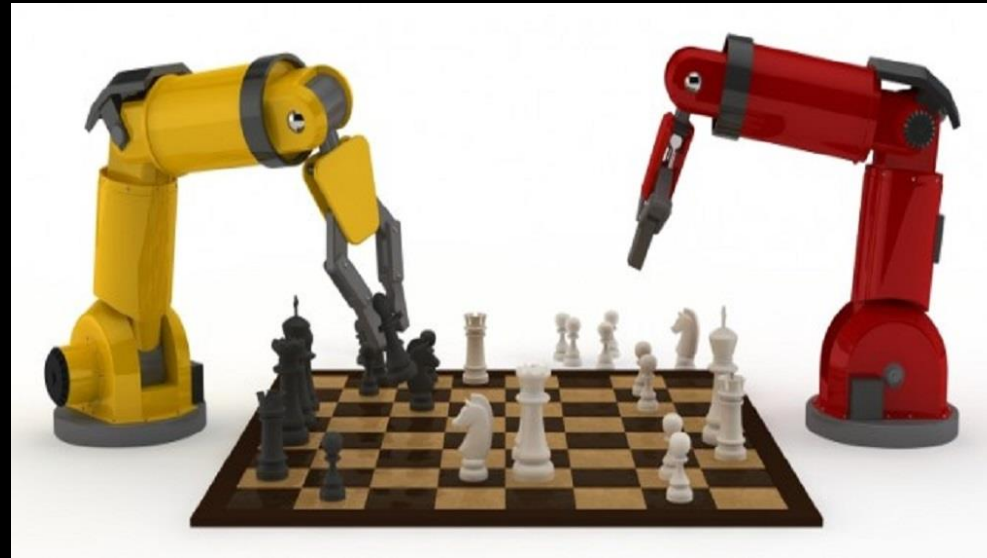
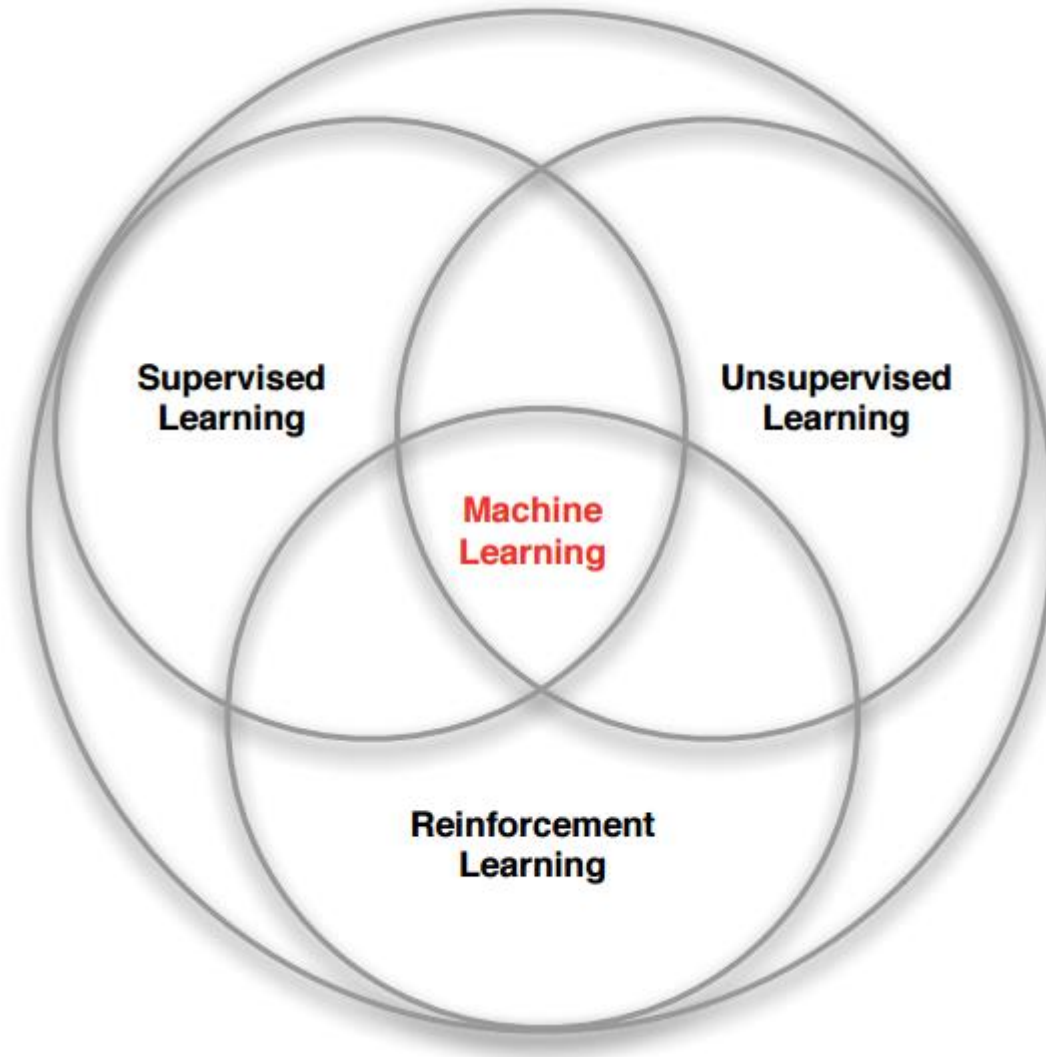


Introduction to Reinforcement Learning

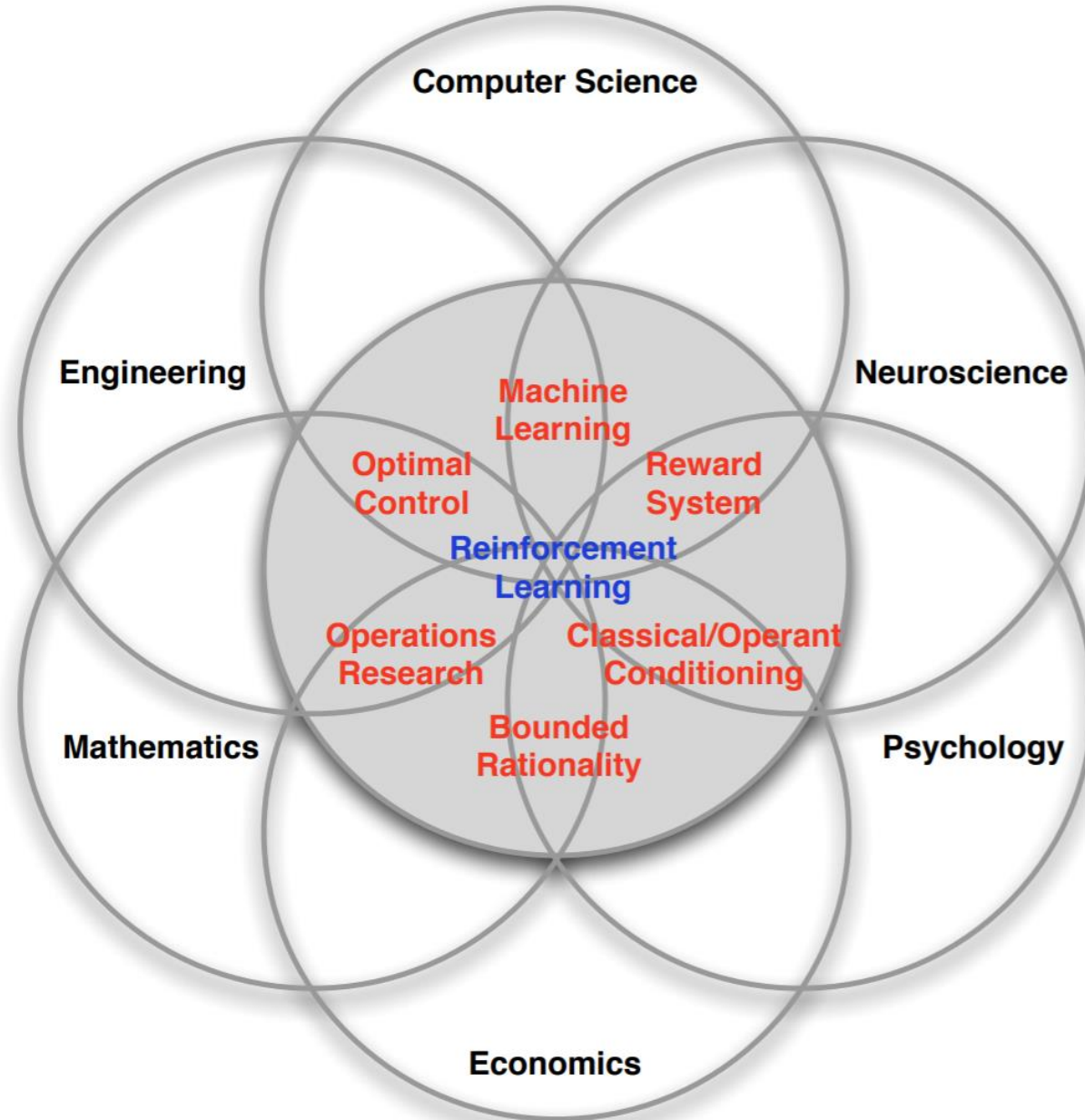


Vladimir Zlobin
vladimir.zlobin@intel.com
CV Summer Camp, July 2020

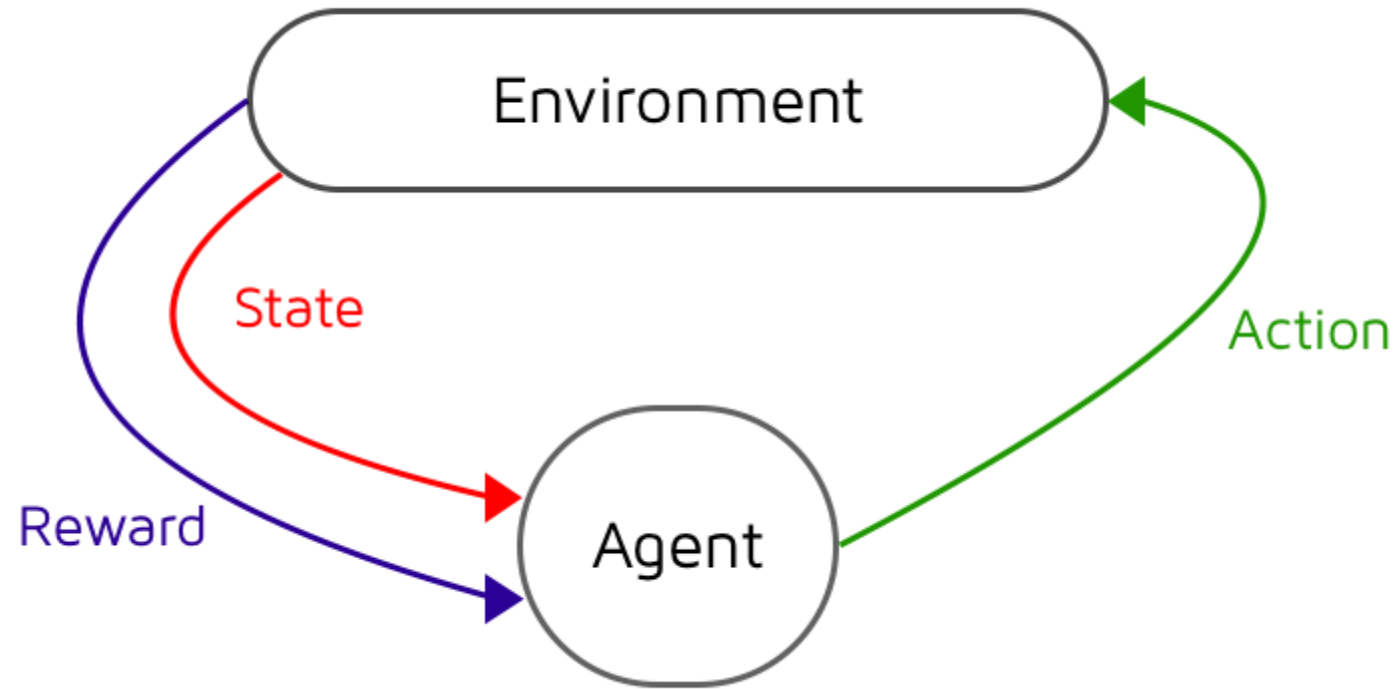
Branches of Machine Learning



Many faces of Reinforcement Learning



RL Mechanics



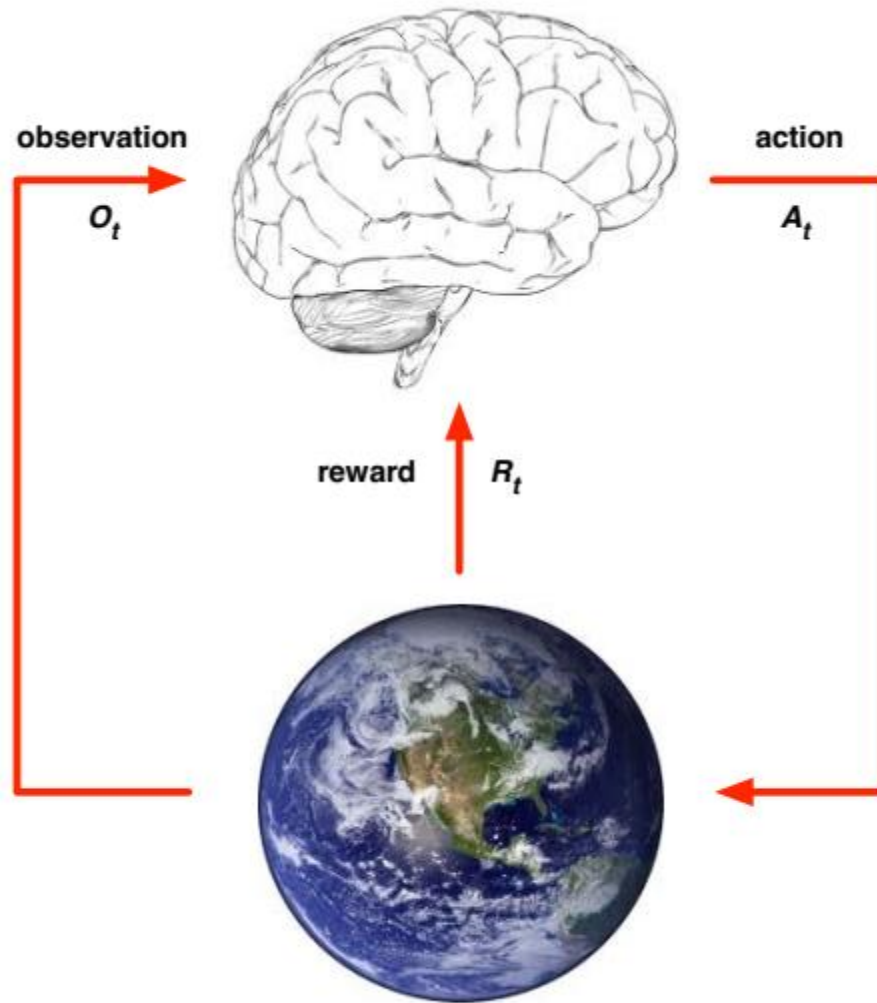
Reinforcement Learning Peculiarities

- What makes RL domain different from other machine learning approaches?
 - There is **no** external supervisor, only *reward* signal
 - Feedback can be delayed – not right now!
 - Time matters – sequential, not i.i.d. data
 - Agent's actions affect future data it receives

Sequential Decision Making

- Goal: *select actions to maximize total future reward*
- Actions may have long term consequences
- Reward may be delayed
- It may be better to sacrifice immediate reward to gain more long-term reward
 - A financial investment (may take months to mature)
 - Refueling a helicopter (might prevent a crash in several hours)
 - Blocking opponent moves (might help winning chances many moves from now)

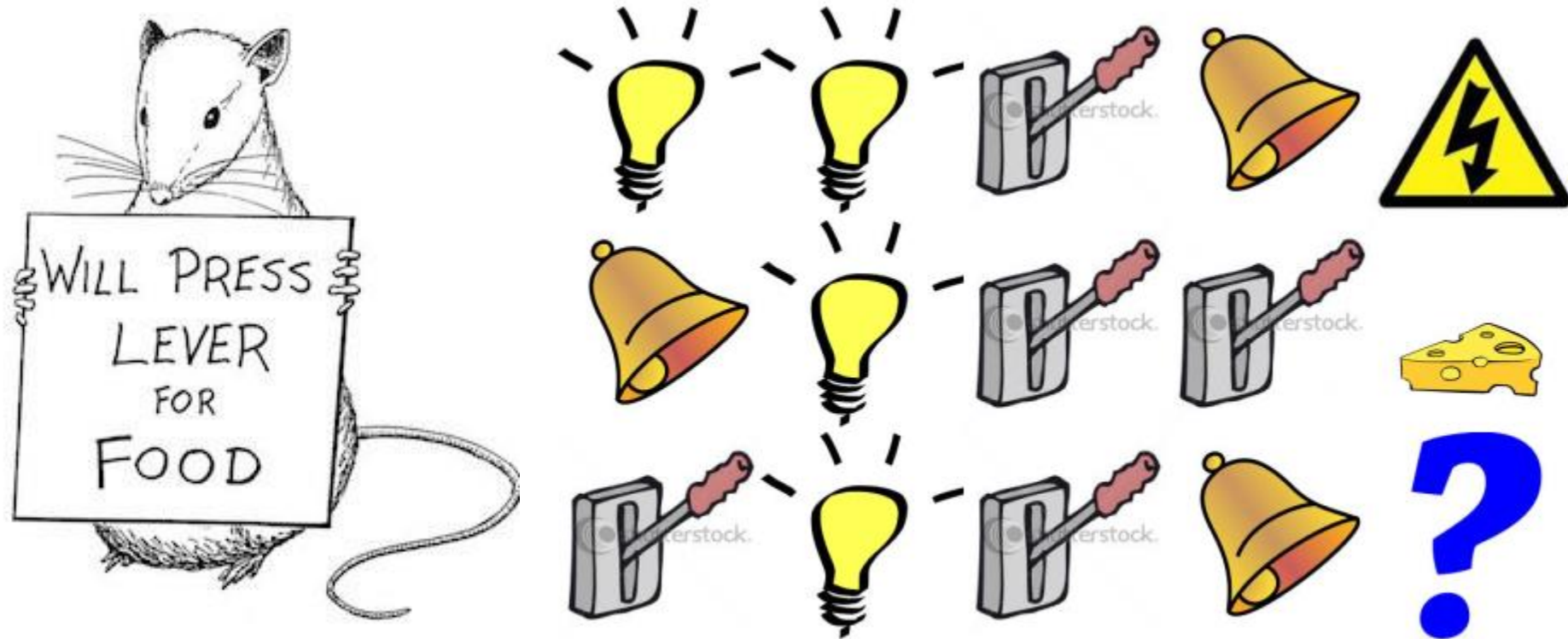
RL Basics



An RL agent may include one or more of these components:

- **Policy:** agent's behavior function
- **Value function:** how good is each state and/or action
- **Model:** agent's representation of the environment

Rat Example



- What if agent state = last 3 items in sequence?
- What if agent state = counts for lights, bells ?
- What if agent state = complete sequence?

Major Components of an RL Agent

- An RL agent may include one or more of these components:
 - Policy: agent's behavior function
 - Value function: how good is each state and/or action
 - Model: agent's representation of the environment

Policy

- A **policy** is the agent's behavior
- It is a map from state to action, e.g.
- Deterministic policy: $a = \pi(s)$
- Stochastic policy: $\pi(a | s) = P[A_t = a | S_t = s]$

Value Function

- Value function is a prediction of future reward
- Used to evaluate the goodness/badness of states
- And therefore to select between actions, e.g.

$$v_{\pi}(s) = E_{\pi}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \mid S_t = s]$$

Model

- A **model** predicts what the environment will do next
- P predicts the next state
- R predicts the next (immediate) reward, e.g.

$$P_{ss'}^a = P[S_{t+1} = s' \mid S_t = s, A_t = a]$$
$$R_s^a = E[R_{t+1} \mid S_t = s, A_t = a]$$

Rewards hypothesis

- A **reward** R_t is a scalar feedback signal
- Indicates how well agent is doing at step t
- The agent's job is to maximize cumulative reward

Reinforcement Learning is based on the **reward hypothesis**.

The reward hypothesis:

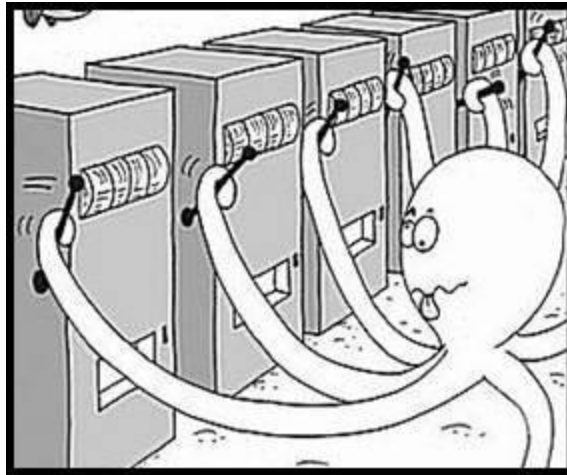
All goals can be described by the maximization of expected cumulative reward.

Exploration & Exploitation

- Reinforcement learning is like *trial-and-error* learning
- The agent should discover a good policy
- From its experiences of the environment
- Without losing too much reward along the way

Exploration & Exploitation

- *Exploration* finds more information about the environment
- *Exploitation* exploits known information to maximize reward
- It is usually important to explore as well as exploit



Examples:

- Bar Selection
 - Exploitation: Go to your favorite bar
 - Exploration: Try a new bar
- Online Banner Advertisements
 - Exploitation: Show the most successful advert
 - Exploration: Show a different advert
- Oil Drilling
 - Exploitation: Drill at the best known location
 - Exploration: Drill at a new location
- Game Playing
 - Exploitation: Play the move you believe is best
 - Exploration: Play an experimental move

Markov Processes Family

- Markov Processes (Markov Chain)
- Markov Reward Processes
- Markov Decision Processes
- Extensions to MDPs:
 - Infinite & Continuous MDP
 - POMDP
 - Undiscounted MDP

Introduction to MDPs

- *Markov decision processes* formally describe an environment for reinforcement learning
- Where the environment is *fully observable*
- i.e. The current *state* completely characterizes the process
- Almost all RL problems can be formalized as MDPs, e.g.
 - Optimal control primarily deals with continuous MDPs
 - Partially observable problems can be converted into MDPs
 - Bandits are MDPs with one state

Markov Property

- Definition: a state S_t is **Markov** if and only if

$$P[S_{t+1} | S_t] = P[S_{t+1} | S_1, \dots, S_t]$$

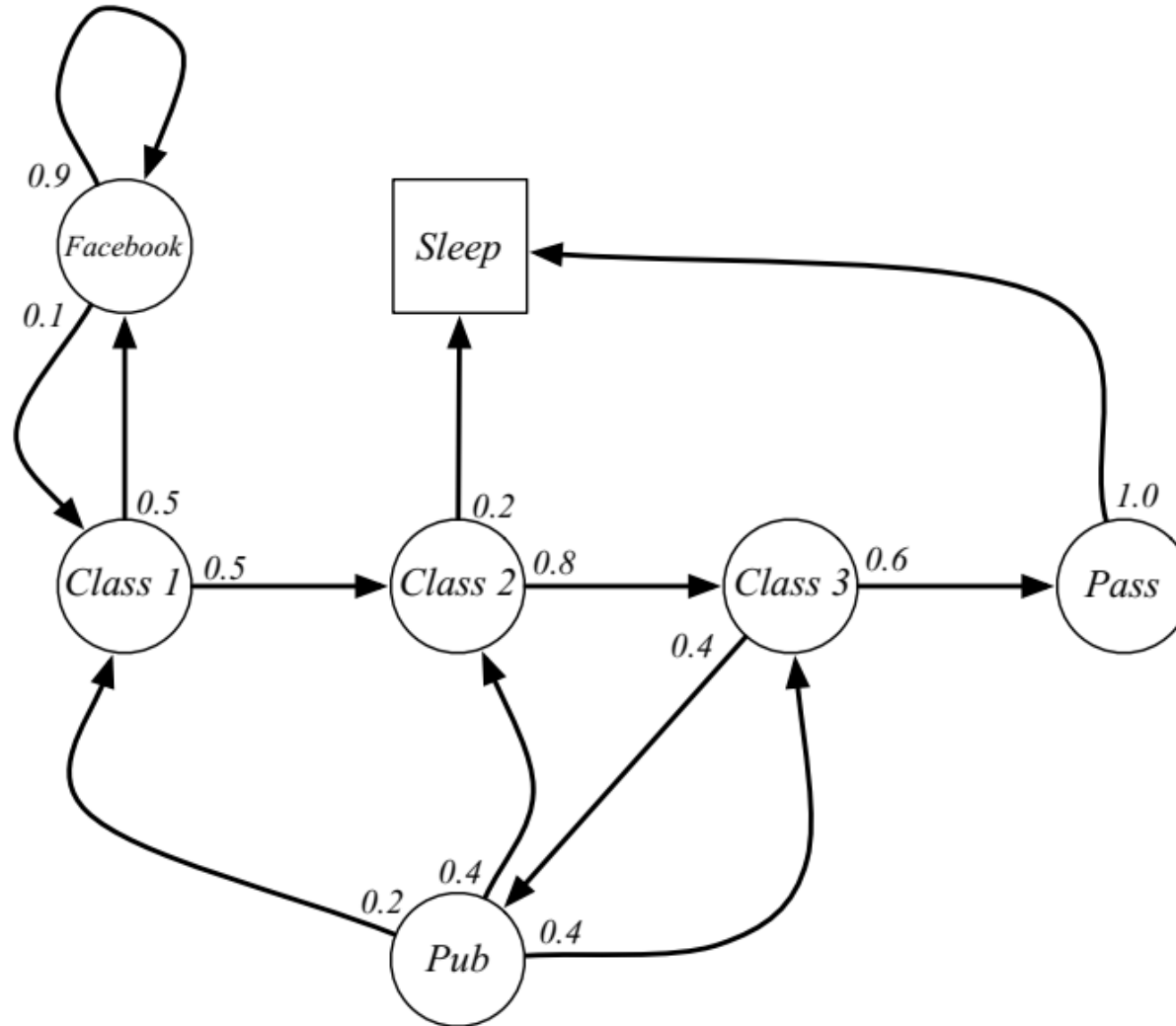
- “The future is independent of the past given the present”
 - The state captures all relevant information from the history
 - Once the state is known, the history may be thrown away
 - i.e. The state is a sufficient statistic of the future

Markov Process

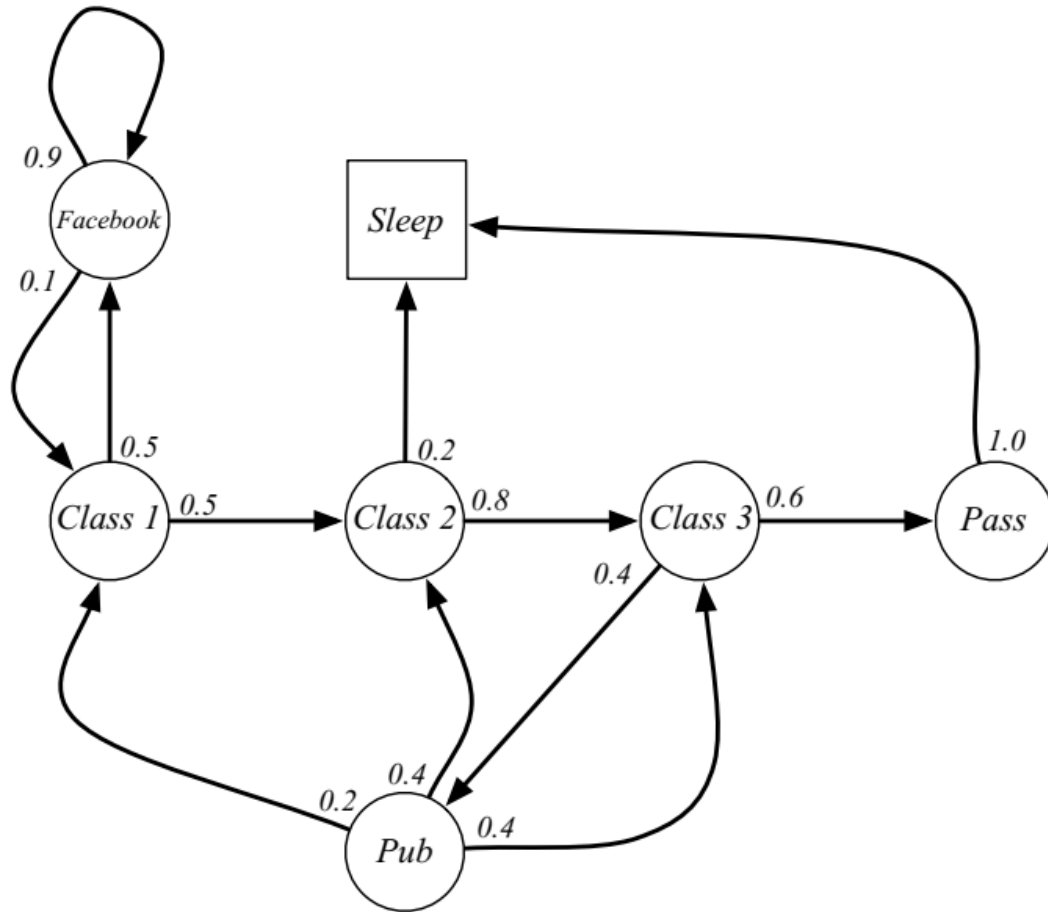
- Markov process is a memoryless random process, i.e. a sequence of random states $\mathbf{S}_1, \dots, \mathbf{S}_t$ with the Markov property.
- *A Markov Process (or Markov Chain)* is a tuple (\mathbf{S}, \mathbf{P})
 - \mathbf{S} is a (finite) set of states
 - \mathbf{P} is a state transition probability matrix
 - $p_{ss'} = P[S_{t+1} = s' \mid S_t = s]$

$$\mathbf{P} = \begin{pmatrix} p_{11} & \cdots & p_{1n} \\ \vdots & \ddots & \vdots \\ p_{n1} & \cdots & p_{nn} \end{pmatrix}$$

Markov Process: Student Example



Markov Process: Episodes Sampling

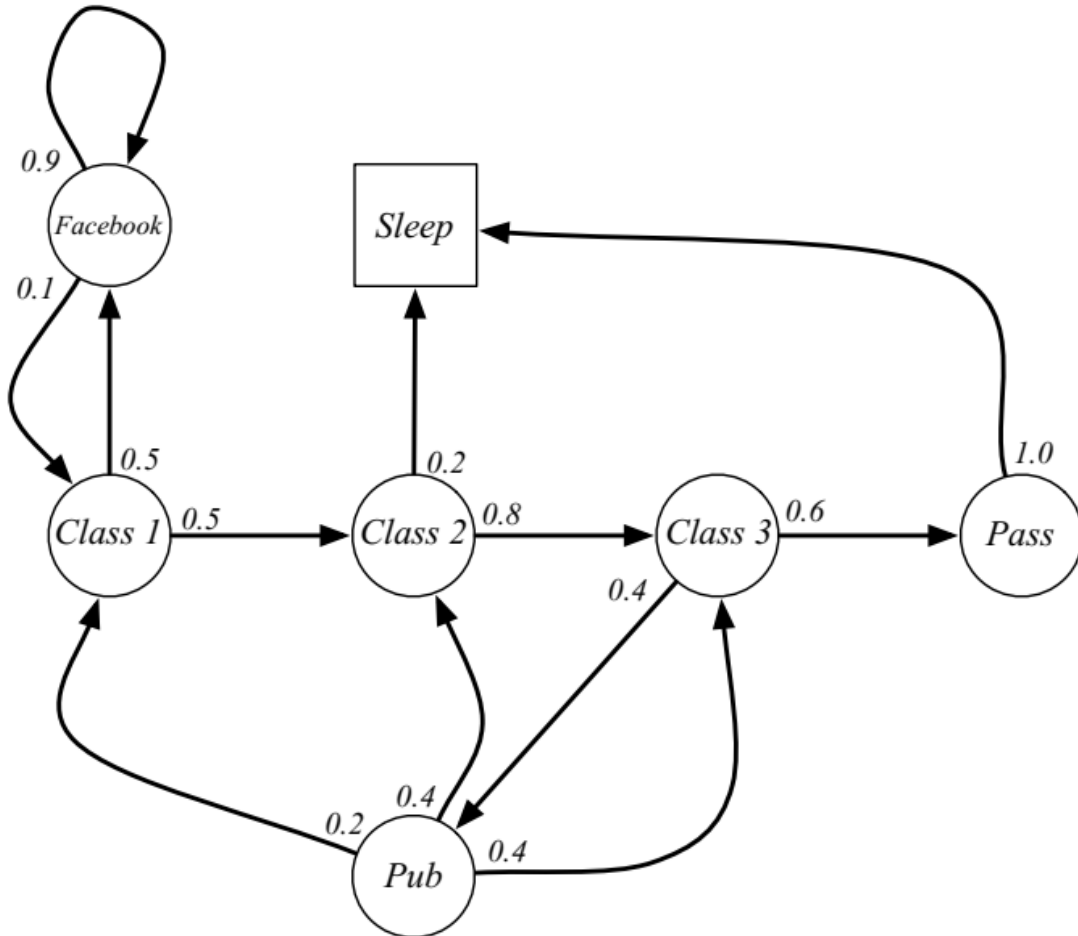


- Sample episodes for Student Markov Chain starting from $S_1 = C1$

S_1, \dots, S_t

- C1 C2 C3 Pass Sleep
- C1 FB FB C1 C2 Sleep
- C1 C2 C3 Pub C2 C3 Pass Sleep
- C1 FB FB C1 C2 C3 Pub C1 FB
FB FB C1 C2 C3 Pub C2 Sleep

Markov Process: Transition Probabilities

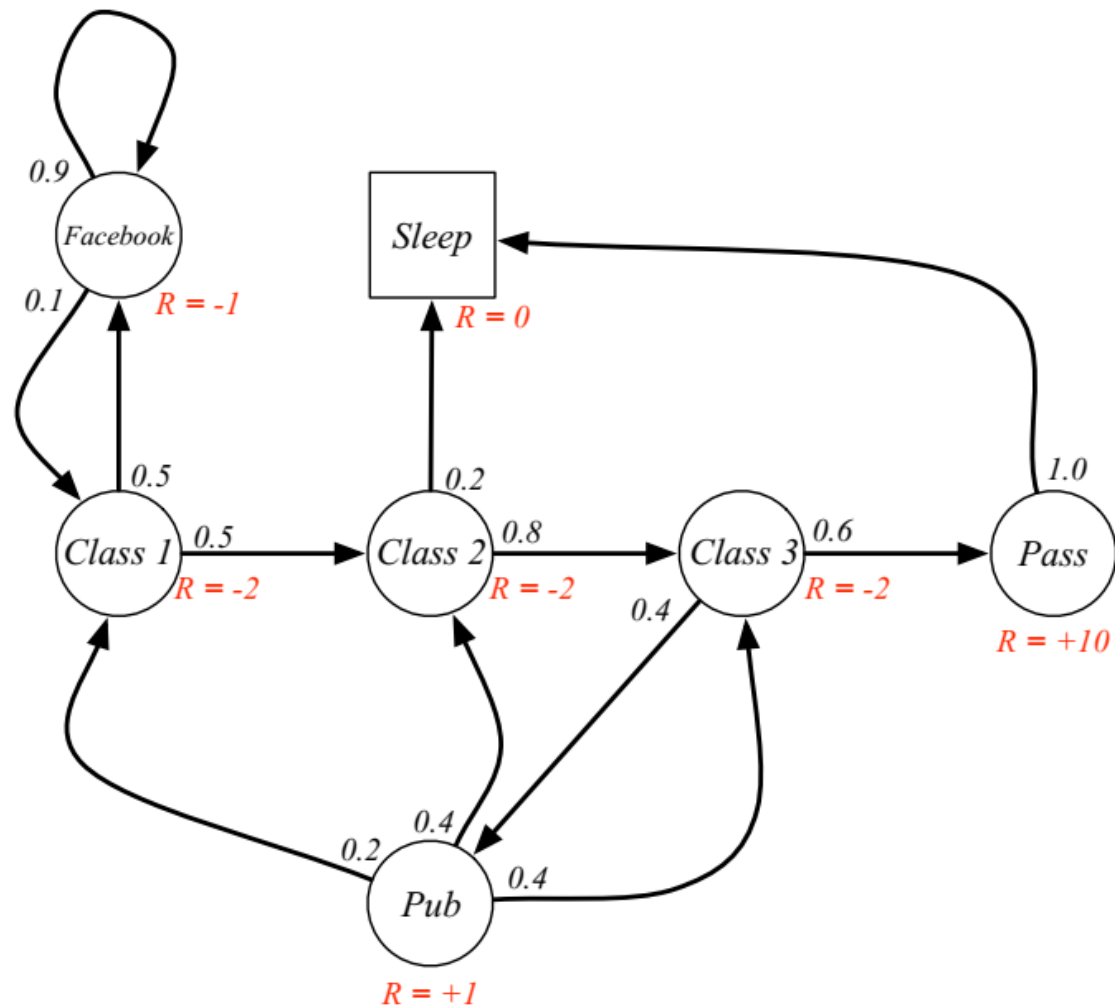


$$\mathcal{P} = \begin{matrix} & \begin{matrix} C1 & C2 & C3 & Pass & Pub & FB & Sleep \end{matrix} \\ \begin{matrix} C1 \\ C2 \\ C3 \\ Pass \\ Pub \\ FB \\ Sleep \end{matrix} & \begin{bmatrix} & & & & & 0.5 & \\ & 0.5 & & & & & 0.2 \\ & & 0.8 & & & & \\ & & & 0.6 & 0.4 & & 1.0 \\ 0.2 & 0.4 & 0.4 & & & & \\ 0.1 & & & & & 0.9 & \\ & & & & & & 1 \end{bmatrix} \end{matrix}$$

Markov Reward Process

- A Markov reward process is a Markov chain with values.
- A *Markov Reward Process* is a tuple (S, P, R, γ)
 - S is a (finite) set of states
 - $p_{ss'}$ is a state transition probability matrix
 - $p_{ss'} = P[S_{t+1} = s' \mid S_t = s]$
 - R is a reward function, $R_s = E[R_{t+1} \mid S_t = s]$
 - γ is a discount factor, $\gamma \in [0, 1]$

MRP: Student Example



Return

- The *return* G_t is the total discounted reward from time-step t .

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+1+k}$$

- The *discount* $\gamma \in [0, 1]$ is the present value of future rewards
- The value of receiving reward R after $k + 1$ time-steps is $\gamma^k R$.
- This values immediate reward above delayed reward.
 - γ close to 0 leads to "myopic" evaluation
 - γ close to 1 leads to "far-sighted" evaluation

Discount intuition

Most Markov reward and decision processes are discounted.
Why?

- Mathematically convenient to discount rewards
- Avoids infinite returns in cyclic Markov processes
- Uncertainty about the future may not be fully represented
- If the reward is financial, immediate rewards may earn more interest than delayed rewards
- Animal/human behavior shows preference for immediate reward
- It is sometimes possible to use *undiscounted* Markov reward processes (i.e. $\gamma=1$), e.g. if all sequences terminate

Value Function

- The value function $v(s)$ gives the long-term value of state s
- The *state value function* $v(s)$ of an MRP is the expected return starting from state s

$$v(s) = E[G_t \mid S_t = s]$$

Student MRP returns

Sample returns for Student MRP with:

- $S_1 = C1$
- $\gamma = 0.5$

$$G_1 = R_2 + \gamma R_3 + \dots + \gamma^{T-2} R_T$$

C1 C2 C3 Pass Sleep

$$v_1 = -2 - 2 * \frac{1}{2} - 2 * \frac{1}{4} + 10 * \frac{1}{8} = -2.25$$

C1 FB FB C1 C2 Sleep

$$v_1 = -2 - 1 * \frac{1}{2} - 1 * \frac{1}{4} - 2 * \frac{1}{8} - 2 * \frac{1}{16} = -3.125$$

C1 C2 C3 Pub C2 C3 Pass Sleep

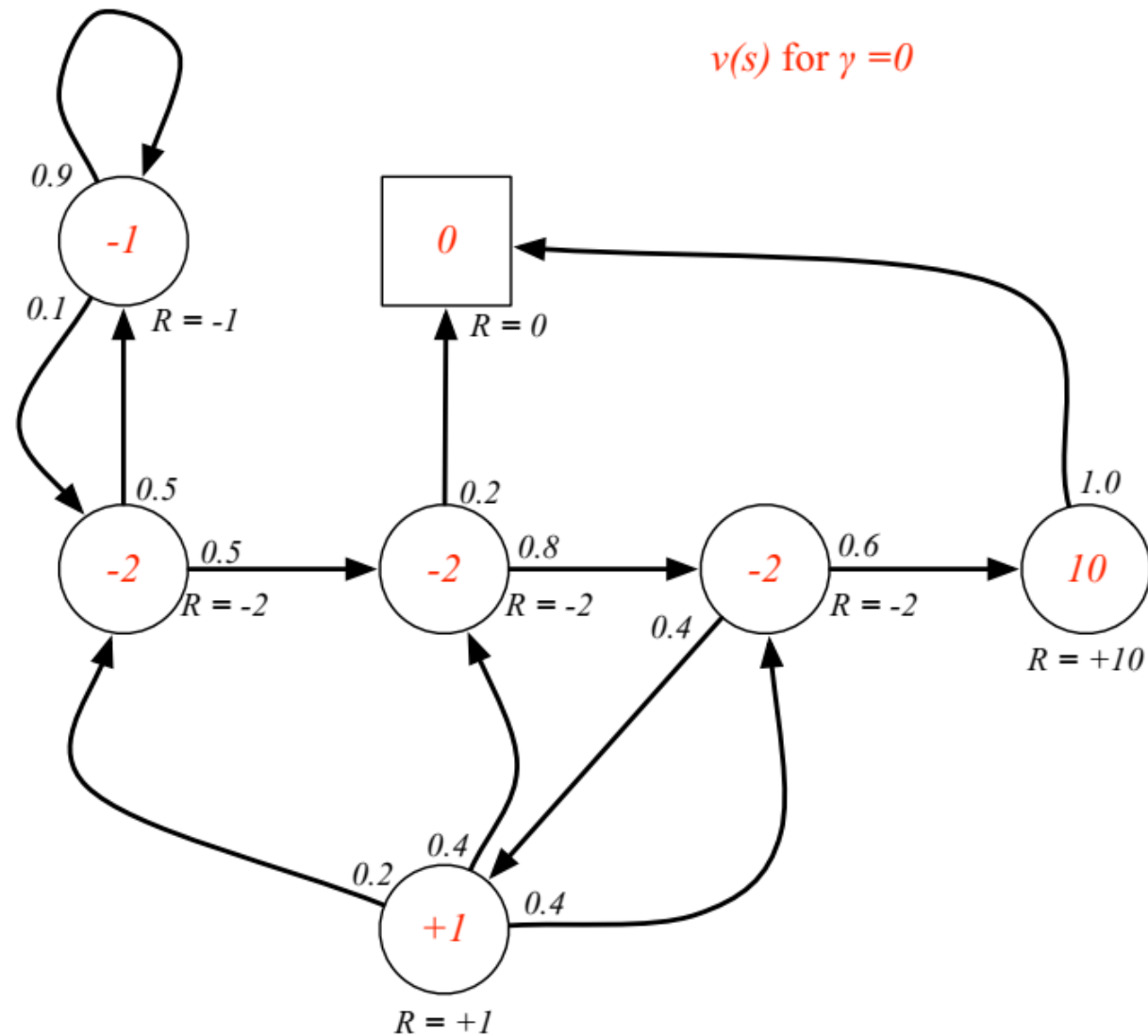
$$v_1 = -2 - 2 * \frac{1}{2} - 2 * \frac{1}{4} + 1 * \frac{1}{8} - 2 * \frac{1}{16} \dots = -3.41$$

C1 FB FB C1 C2 C3 Pub C1 ...

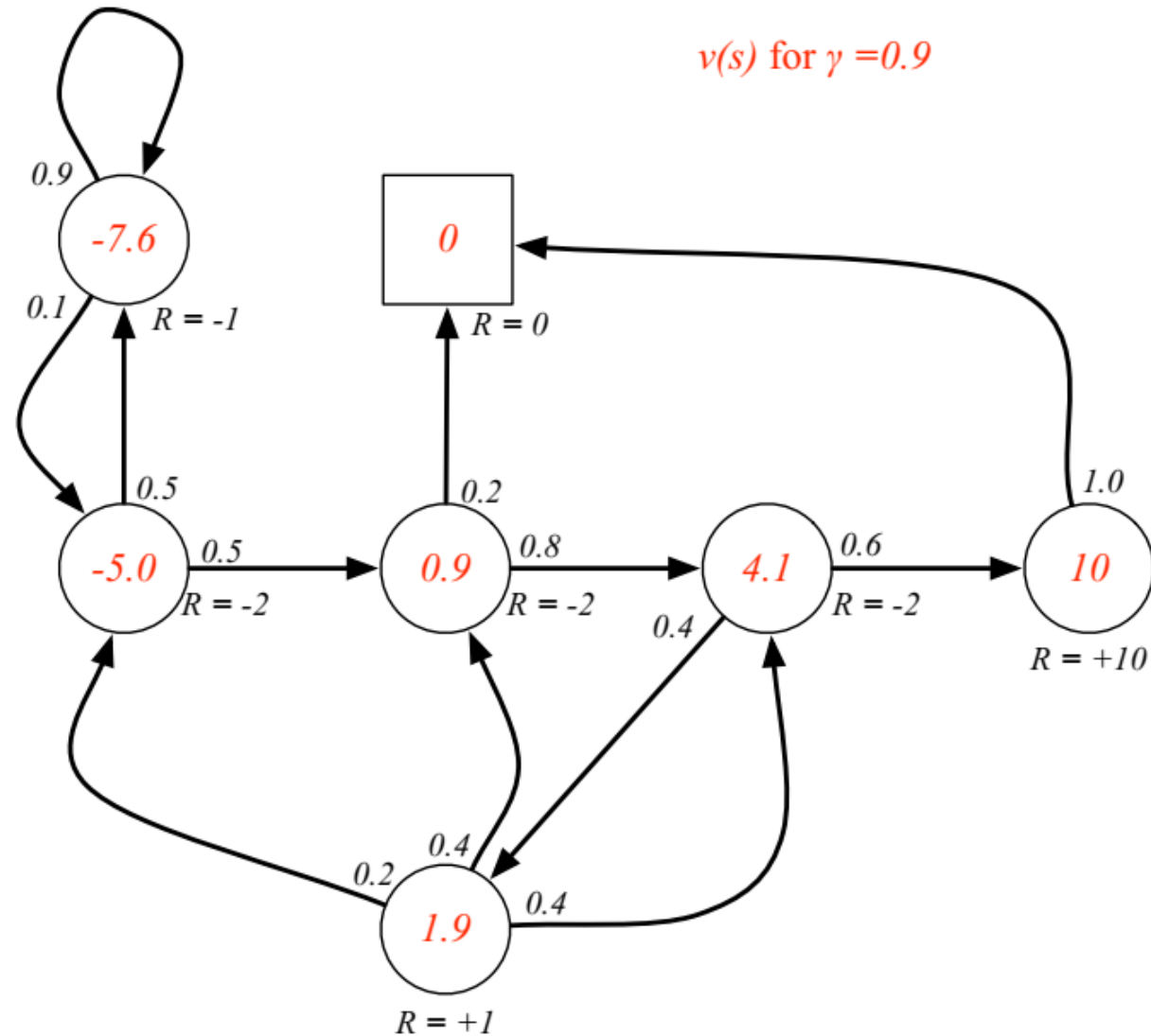
$$v_1 = -2 - 1 * \frac{1}{2} - 1 * \frac{1}{4} - 2 * \frac{1}{8} - 2 * \frac{1}{16} \dots = -3.20$$

FB FB FB C1 C2 C3 Pub C2 Sleep

State-Value function for student MRP



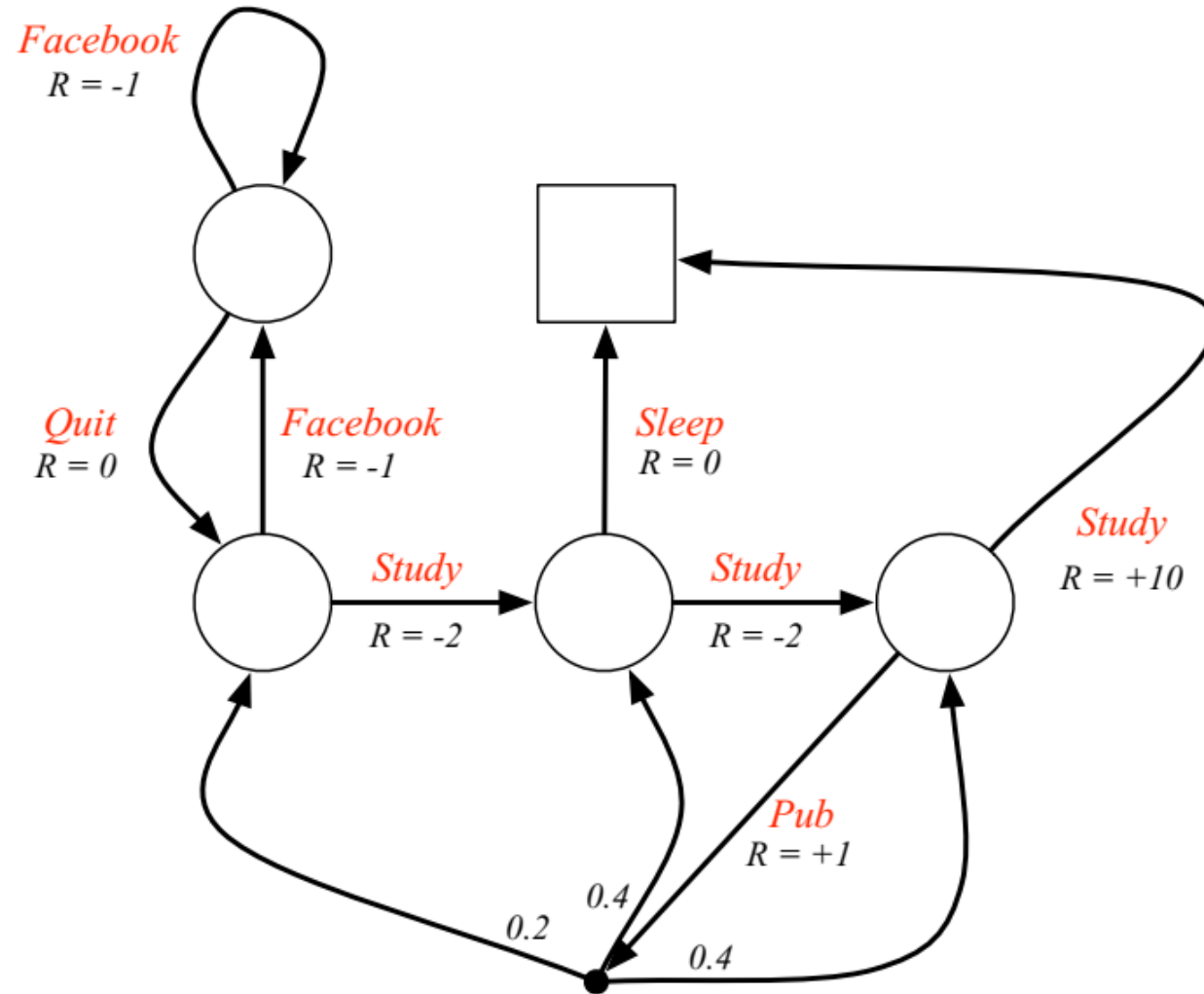
State-Value function for student MRP



Markov Decision Process

- A Markov decision process (MDP) is a Markov reward process with decisions. It is an *environment* in which all states are Markov.
- A Markov reward process is a Markov chain with values.
- A *Markov Decision Process* is a tuple (S, A, P, R, γ)
 - S is a (finite) set of states
 - A is a finite set of actions
 - $p^a_{ss'}$ is a state transition probability matrix
 - $p^a_{ss'} = P[S_{t+1} = s' \mid S_t = s, A_t = a]$
 - R is a reward function, $R^a_s = E[R_{t+1} \mid S_t = s, A_t = a]$
 - γ is a discount factor, $\gamma \in [0, 1]$

Markov Decision Process: Example



MDP Policies

A *policy* π is a distribution over actions given states

$$\pi(a \mid s) = P[A_t = a \mid S_t = s]$$

- A policy fully defines the behavior of an agent
- MDP policies depend on the current state (not the history)
- i.e. Policies are *stationary* (time-independent),

$$A_t \sim \pi(\cdot \mid S_t); \forall t > 0$$

Updated Value functions

- The *state-value function* $v_{\pi}(s)$ of an MDP is the expected return starting from state s , and then following policy π

$$v_{\pi}(s) = E_{\pi} [G_t \mid S_t = s]$$

- The *action-value function* $q_{\pi}(s, a)$ is the expected return starting from state s , taking action a , and then following policy π

$$q_{\pi}(s, a) = E_{\pi} [G_t \mid S_t = s, A_t = a]$$

Optimal Value Function

- The *optimal state-value function* $v_*(s)$ is the maximum value function over all policies

$$v_*(s) = \max_{\pi} (v_{\pi}(s))$$

- The *optimal action-value function* $q_*(s; a)$ is the maximum action-value function over all policies

$$q_*(s, a) = \max_{\pi} (q_{\pi}(s, a))$$

- The optimal value function specifies the best possible performance in the MDP.
- An MDP is “solved” when we know the optimal value function.

Optimal Policy

- Define a partial ordering over policies:

$$\pi \geq \pi' \text{ if } v_{\pi}(s) \geq v_{\pi'}(s) \quad \forall s$$

Theorem: For any Markov Decision Process

- There exists an optimal policy π^* that is better than or equal to all other policies: $\pi^* \geq \pi \quad \forall \pi$
- All optimal policies achieve the optimal value function,

$$v_{\pi^*}(s) = v^*(s)$$

- All optimal policies achieve the optimal action-value function

$$q_{\pi^*}(s, a) = q^*(s, a)$$

Recap: Dynamic Programming

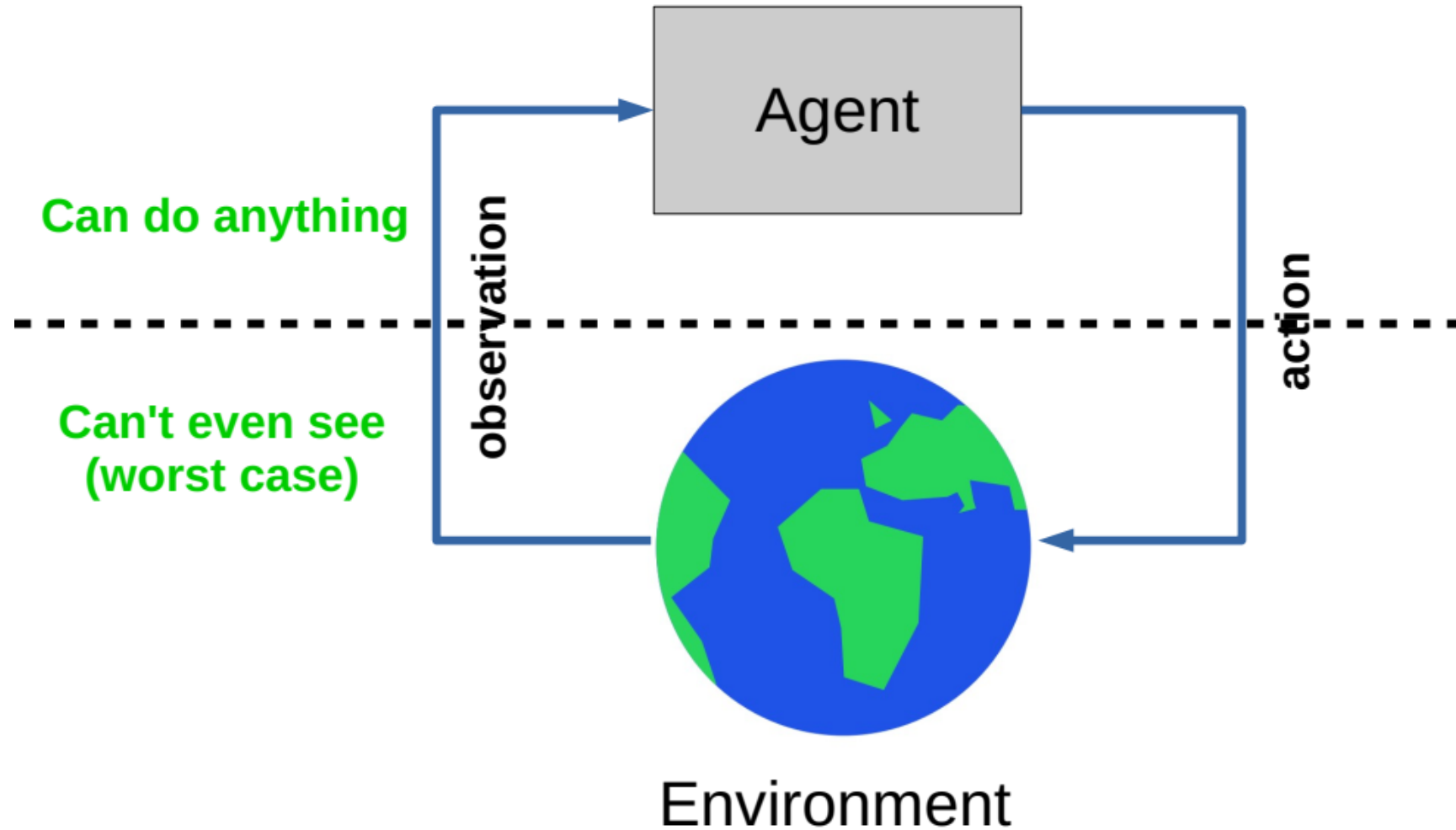
- $v_{\pi}(s), v_{*}(s)$
- If you know $v_{*}(s), p(r, s' | s, a) \rightarrow$ know optimal policy
- We can learn $v_{*}(s)$ with Dynamic Programming:

$$v_{*}(s) = \max_a \sum_{r, s'} p(r, s' | s, a) [r + \gamma v_{*}(s')]$$

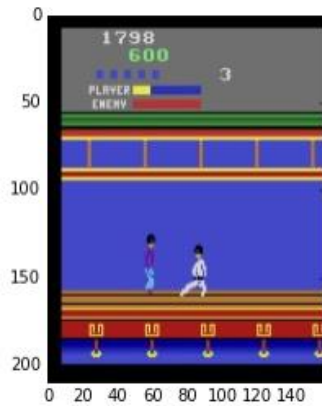
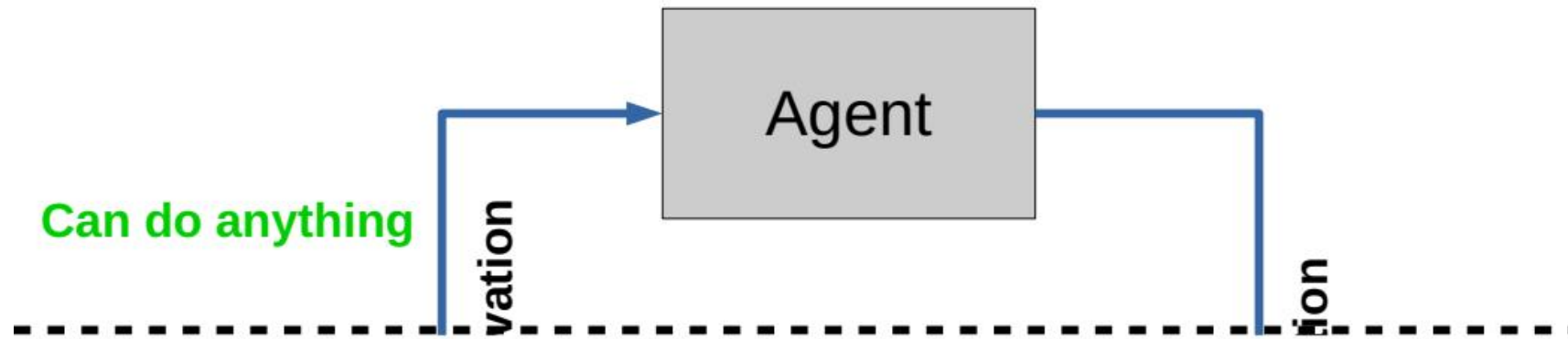
- $q_{\pi}(s, a), q_{*}(s, a)$

$$q_{*}(s, a) = \sum_{r, s'} p(r, s' | s, a) [r + \gamma \max_{a'} q_{*}(s', a')]]$$

Decision making: reality check



Decision making: reality check



Model-Free Setup

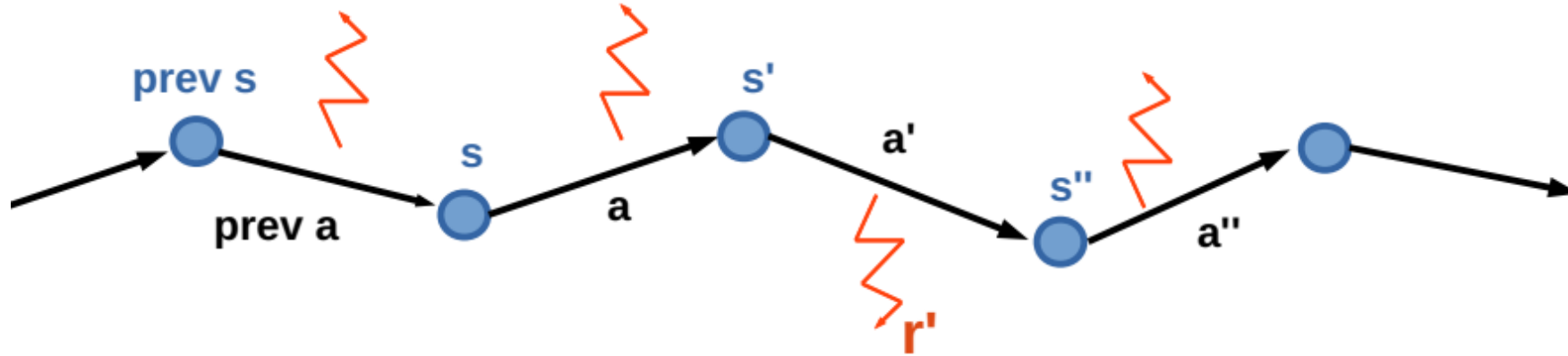
- We don't know internal environment representation, e.g.

$$p(r, s' \mid s, a) - \text{unknown}$$

What should we do?

Learning from trajectories

- $s_1 \rightarrow a_1 \rightarrow r_1 \rightarrow s_2 \rightarrow \dots \rightarrow s_n$ – trajectory



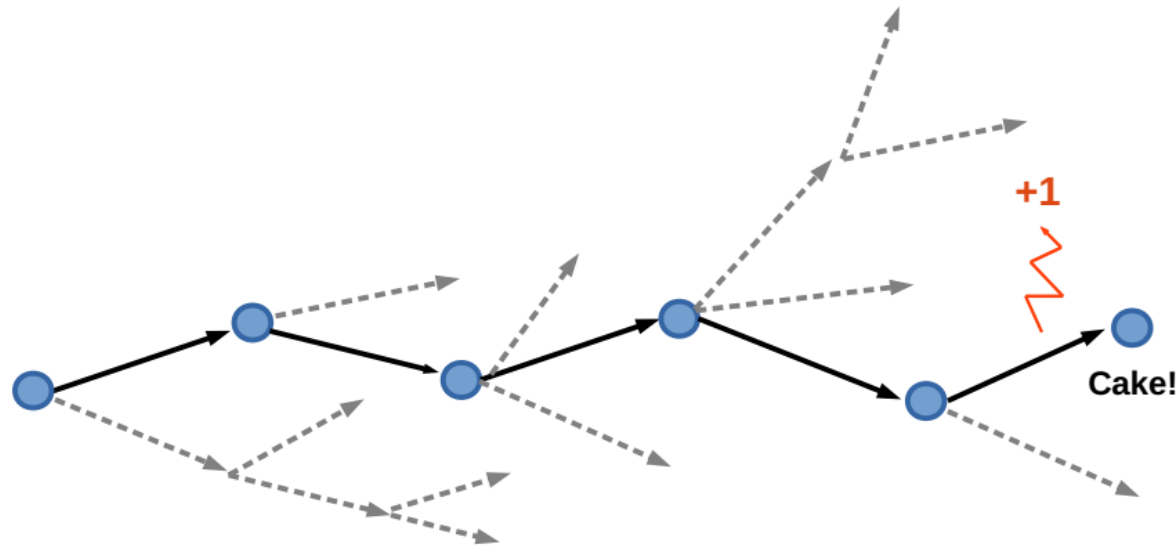
- Model-based setup:
 - you can apply Dynamic Programming
 - you can plan (!)
- Model-free setup:
 - you can experiment with different actions
 - no guaranties (!!!)

Learning from trajectories

- $s_1 \rightarrow a_1 \rightarrow r_1 \rightarrow s_2 \rightarrow \dots \rightarrow s_n$ – trajectory
- We can sample trajectories (a lot of trajectories!)
- What should we learn ?
 - $p(r, s' | s, a)$
 - $v_\pi(s)$
 - $q_\pi(s, a)$

Monte-Carlo RL

- Just like N+1 heuristic:
 - Get all trajectories containing particular (s, a)
 - Estimate $G_t(s, a)$ for each trajectory
 - Average them to get *estimation* of expectation



Monte-Carlo RL

- MC methods learn directly from episodes of experience
- MC is *model-free*: no knowledge of MDP transitions / rewards
- MC learns from *complete* episodes: no bootstrapping
- MC uses the simplest possible idea: value = mean return
- Note: can only apply MC to *episodic* MDPs
 - All episodes must terminate

Temporal Difference Learning

- TD methods learn directly from episodes of experience
- TD is *model-free*: no knowledge of MDP transitions / rewards
- TD learns from *incomplete* episodes, by *bootstrapping*
- TD updates a guess towards a guess

Temporal Difference

- Just like in the 'incremental mean' example we can improve $q_{\pi}(s, a)$ iteratively:

$$q_*(s, a) = \sum_{r, s'} p(r, s' | s, a) [r + \gamma \max_{a'} q_*(s', a')]]$$

Temporal Difference

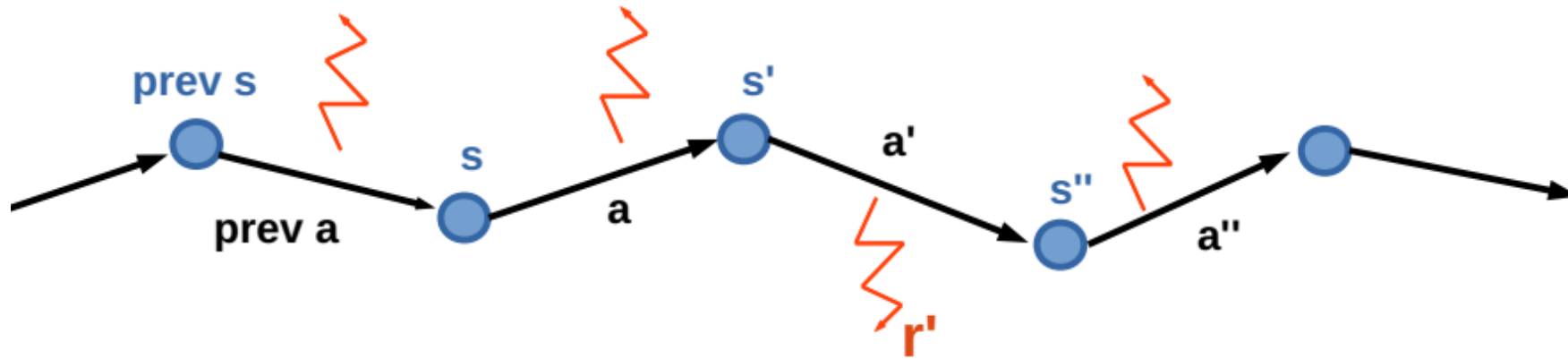
$$\sum_{r,s'} p(r, s' \mid s, a) [r + \gamma \max_{a'} q_*(s', a')] \approx$$

$$\approx \frac{1}{N} \sum_i r_i + \gamma \max_{a'} Q(s'_i, a')$$

- One more trick: use incremental averaging with 1 sample

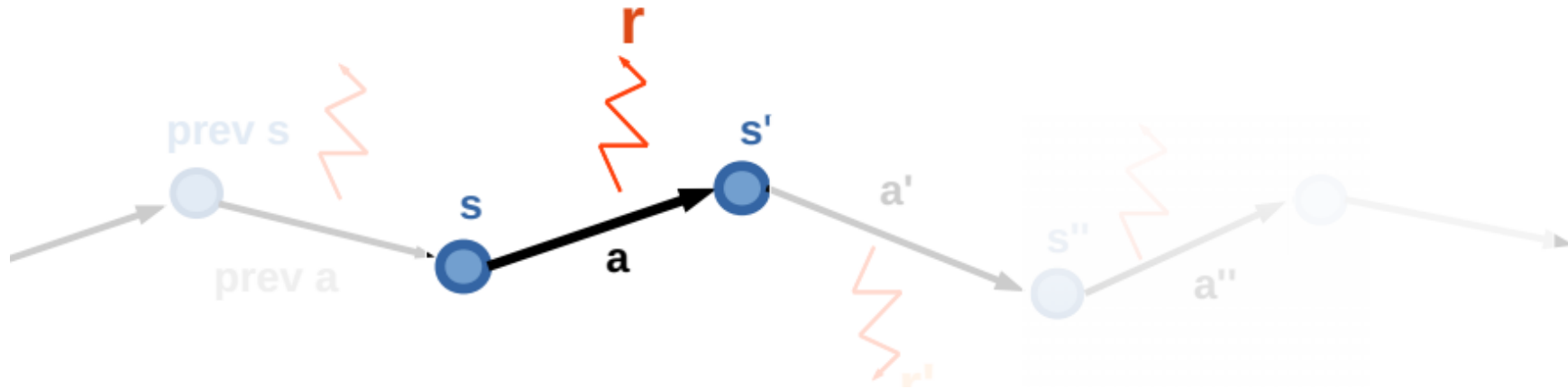
$$Q(s, a) = \alpha (r_t + \gamma \max_{a'} Q(s', a')) + (1 - \alpha) Q(s, a)$$

Q-learning



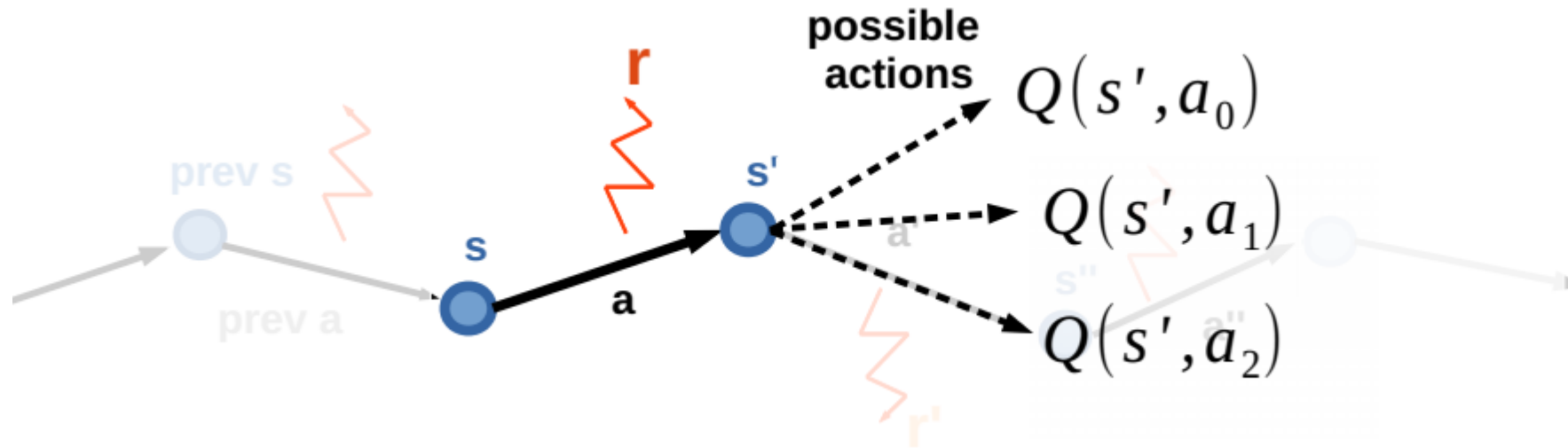
- Works on a sequence of
 - states (s)
 - actions (a)
 - rewards (r)

Q-learning



- Initialize $Q(s, a)$ with zeros
- Cycle:
 - Sample $\langle s, a, r, s' \rangle$ from environment

Q-learning



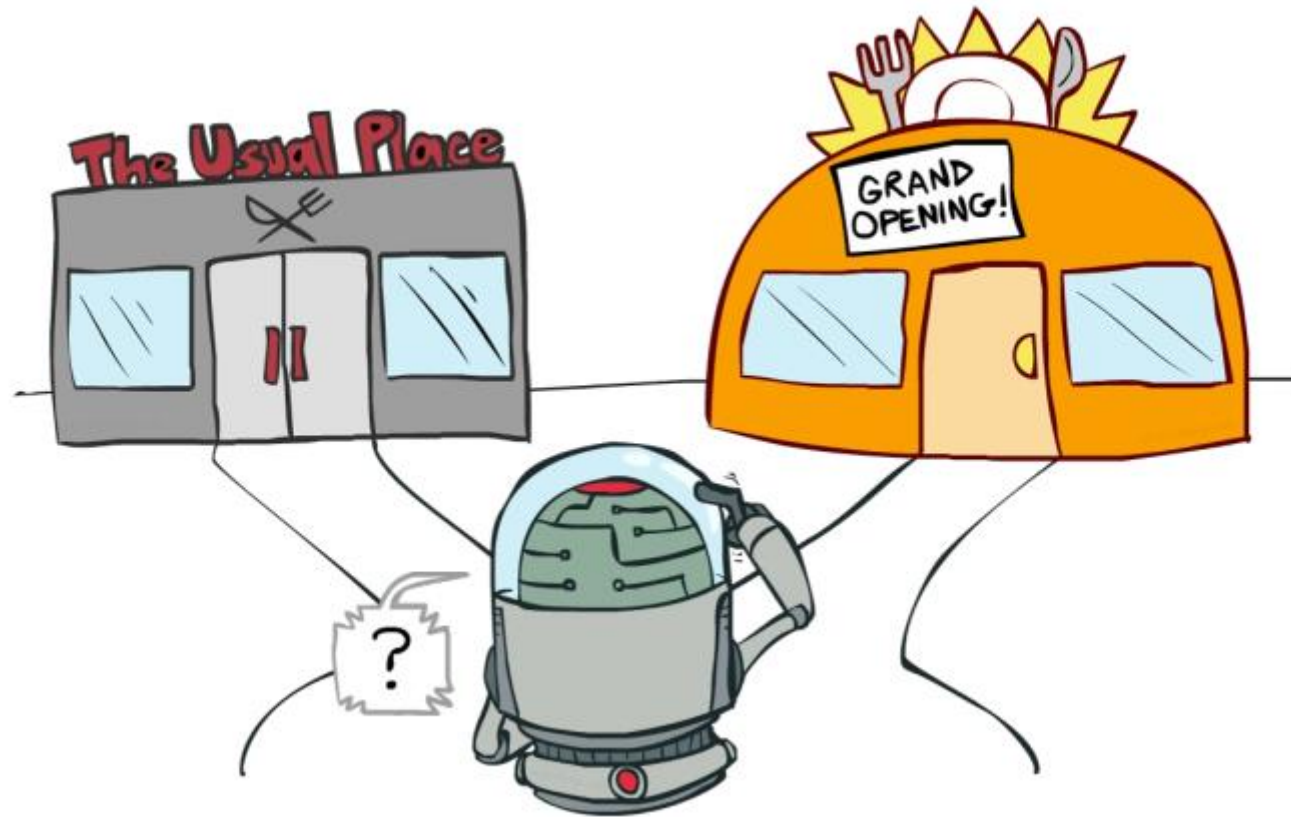
- Initialize $Q(s, a)$ with zeros
- Cycle:
 - Sample $\langle s, a, r, s' \rangle$ from environment
 - Compute $\hat{Q}(s, a) = r + \gamma \max_{a_i} Q(s', a_i)$
 - Update: $Q(s, a) = \alpha \hat{Q}(s, a) + (1 - \alpha) Q(s, a)$

MC vs TD

- TD can learn *before* knowing the final outcome
 - TD can learn online after every step
 - MC must wait until end of episode before return is known
- TD can learn *without* the final outcome
 - TD can learn from incomplete sequences
 - MC can only learn from complete sequences
 - TD works in continuing (non-terminating) environments
 - MC only works for episodic (terminating) environments

Exploration/Exploitation Revisited

- Balance between using what you learned and trying to find something even better



Exploration/Exploitation Revisited

- Strategies:

- ϵ -greedy

With probability ϵ take random action,
otherwise take optimal action.

- Softmax

Pick action proportional to softmax Q-values.

$$\pi(a \mid s) = \text{softmax}(Q(s,a))$$

- ϵ - dithering

Adding random noise to Q-values with ϵ probability

Exploration/Exploitation over time

- If you want to converge to optimal policy you need to gradually reduce exploration.
- Example:
Initialize ϵ -greedy $\epsilon = 0.5$, then gradually reduce it
 - If $\epsilon \rightarrow 0$, it's **greedy in the limit**

Reinforcement Learning in the Wild

- State space is usually large, sometimes continuous.
 - Backgammon: **10^{20} states**
 - Computer Go: **10^{170} states**
 - Helicopter: **continuous state space**
- However, states do have a structure, similar states have similar action outcomes

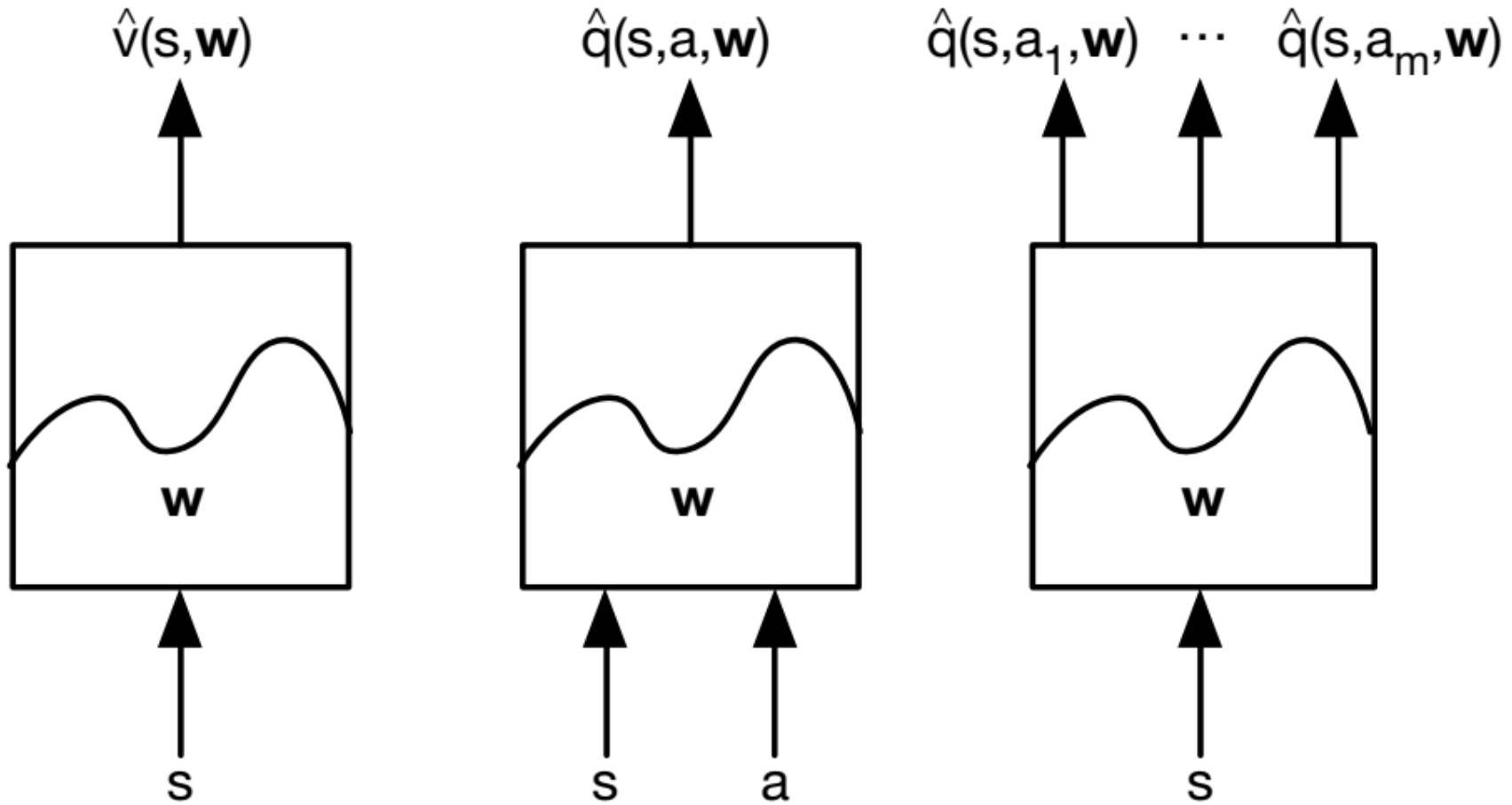
What are we going to do?

**THE SAME THING WE
DO EVERY NIGHT, PINKY**



APPROXIMATE!

Types of Value Function Approximation



Which class of function to choose?

- There are many function approximators, e.g.
 - *Linear combinations of features*
 - *Neural network*
 - Decision tree
 - Nearest neighbor
 - Fourier / wavelet bases
 - ...

Gradient Descent

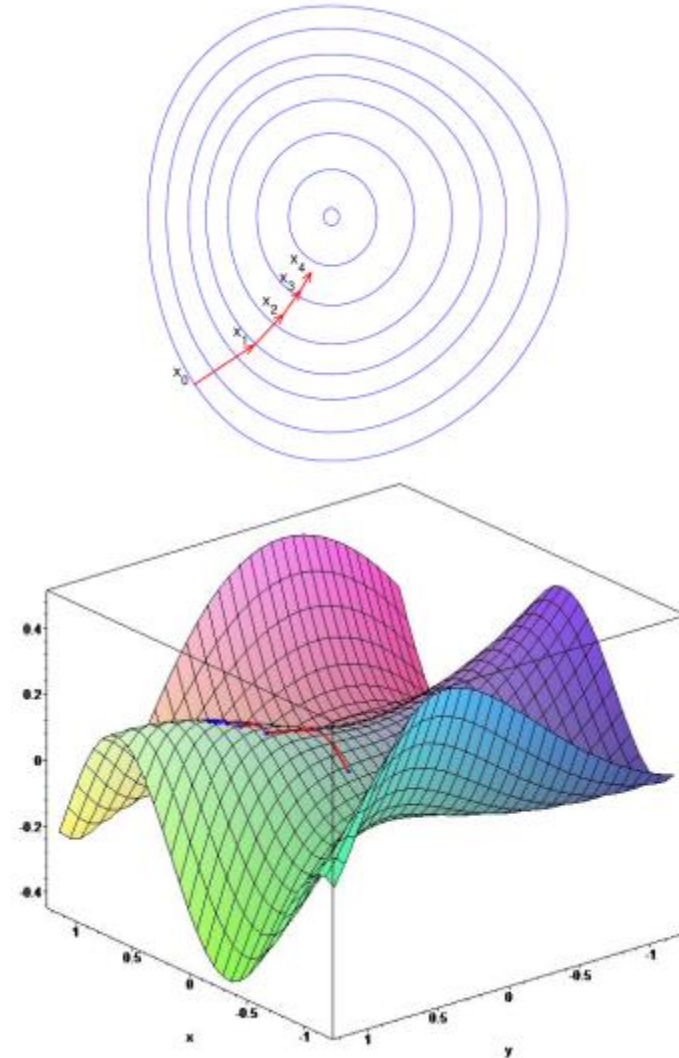
- Let $J(\mathbf{w})$ be a differentiable function of parameter vector \mathbf{w}
- Define the *gradient* of $J(\mathbf{w})$ to be

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \begin{pmatrix} \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_1} \\ \vdots \\ \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_n} \end{pmatrix}$$

- To find a local minimum of $J(\mathbf{w})$:
 - Adjust \mathbf{w} in direction of -gradient

$$\Delta \mathbf{w} = -\frac{1}{2} \alpha \nabla_{\mathbf{w}} J(\mathbf{w})$$

where α is a step-size parameter



SGD for Value Function approximation

- Goal: find parameter vector \mathbf{w} minimizing mean-squared error between approximate value $v'(s, \mathbf{w})$ and true value $v_\pi(s)$

$$J(\mathbf{w}) = \mathbb{E}_\pi [(v_\pi(S) - \hat{v}(S, \mathbf{w}))^2]$$

- Gradient descent finds a local minimum

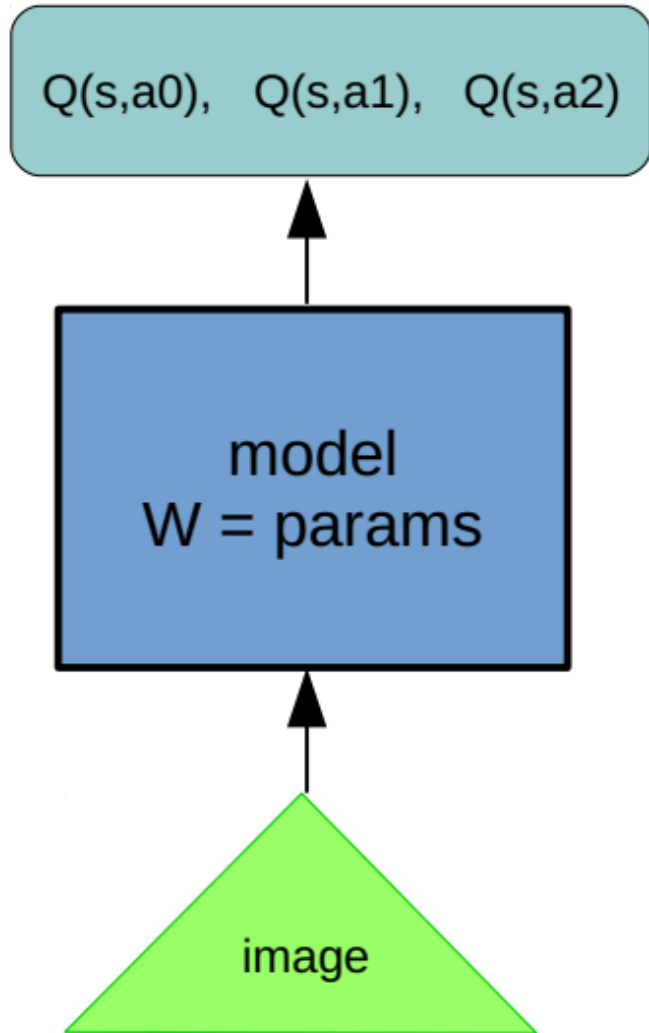
$$\begin{aligned}\Delta \mathbf{w} &= -\frac{1}{2}\alpha \nabla_{\mathbf{w}} J(\mathbf{w}) \\ &= \alpha \mathbb{E}_\pi [(v_\pi(S) - \hat{v}(S, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w})]\end{aligned}$$

- Stochastic gradient descent *samples* the gradient

$$\Delta \mathbf{w} = \alpha (v_\pi(S) - \hat{v}(S, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w})$$

- Expected update is equal to full gradient update

Approximate Q-learning

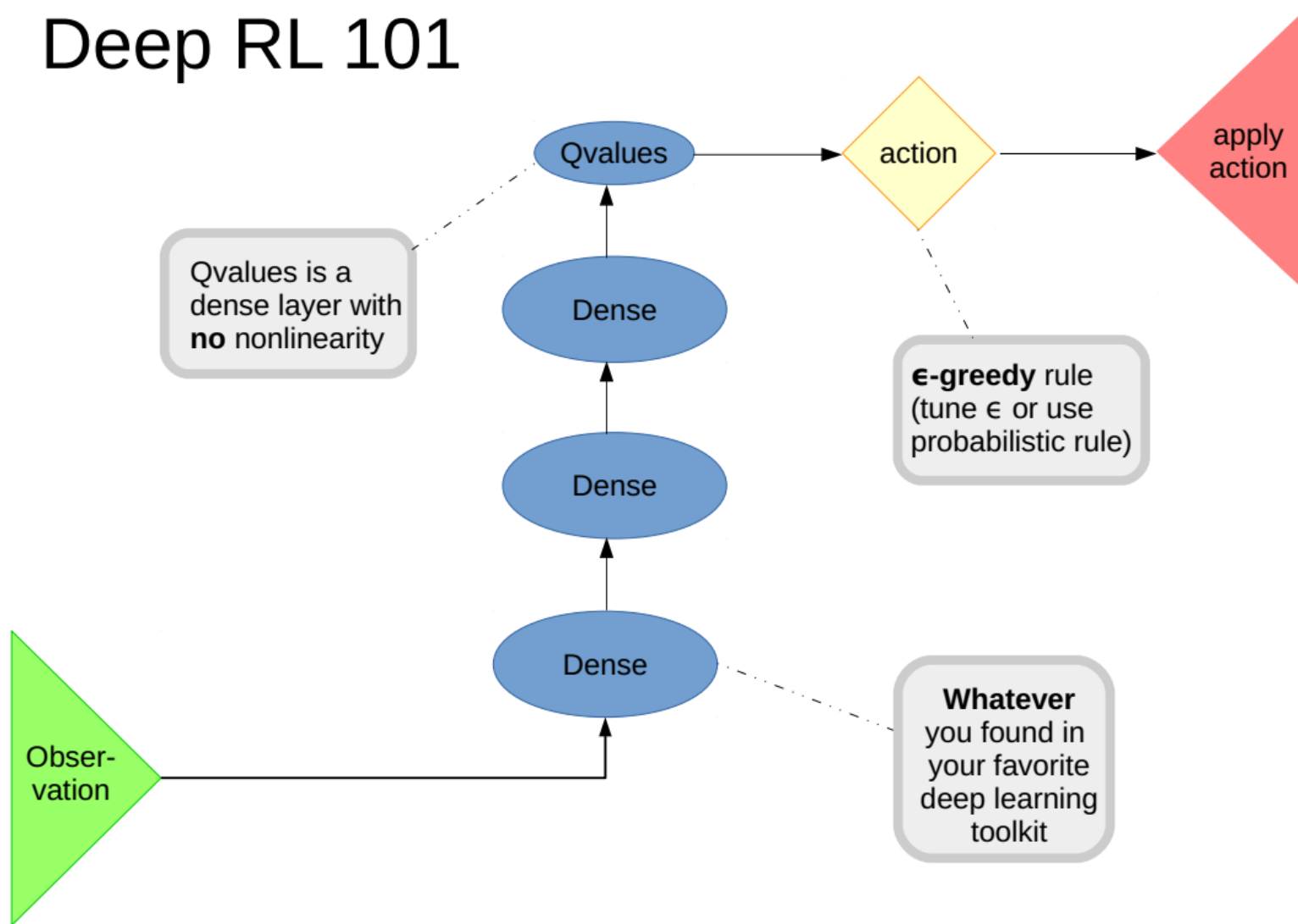


- Initialize W .
- Cycle:
 - Sample $\langle s, a, r, s' \rangle$ from environment
 - Compute $\hat{Q}(s, a) = r + \gamma \max_{a_i} Q(s', a_i)$
 - Objective:
$$L = [Q(s_t, a_t) - \hat{Q}(s_t, a_t)]^2$$
 - SGD Update:

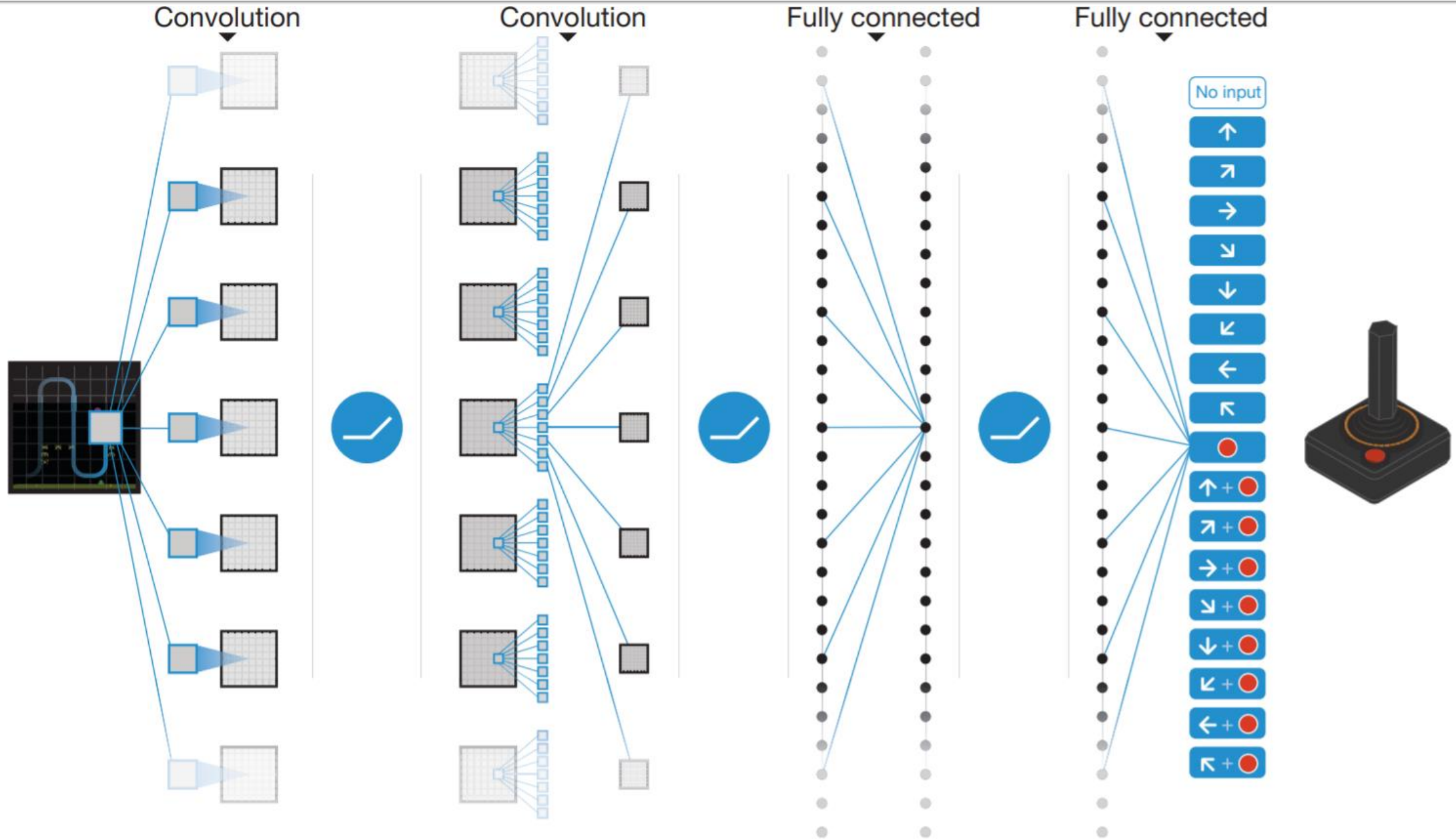
$$W_{t+1} = W_t - \alpha \frac{\partial L}{\partial w_t}$$

RL Mechanics

Deep RL 101

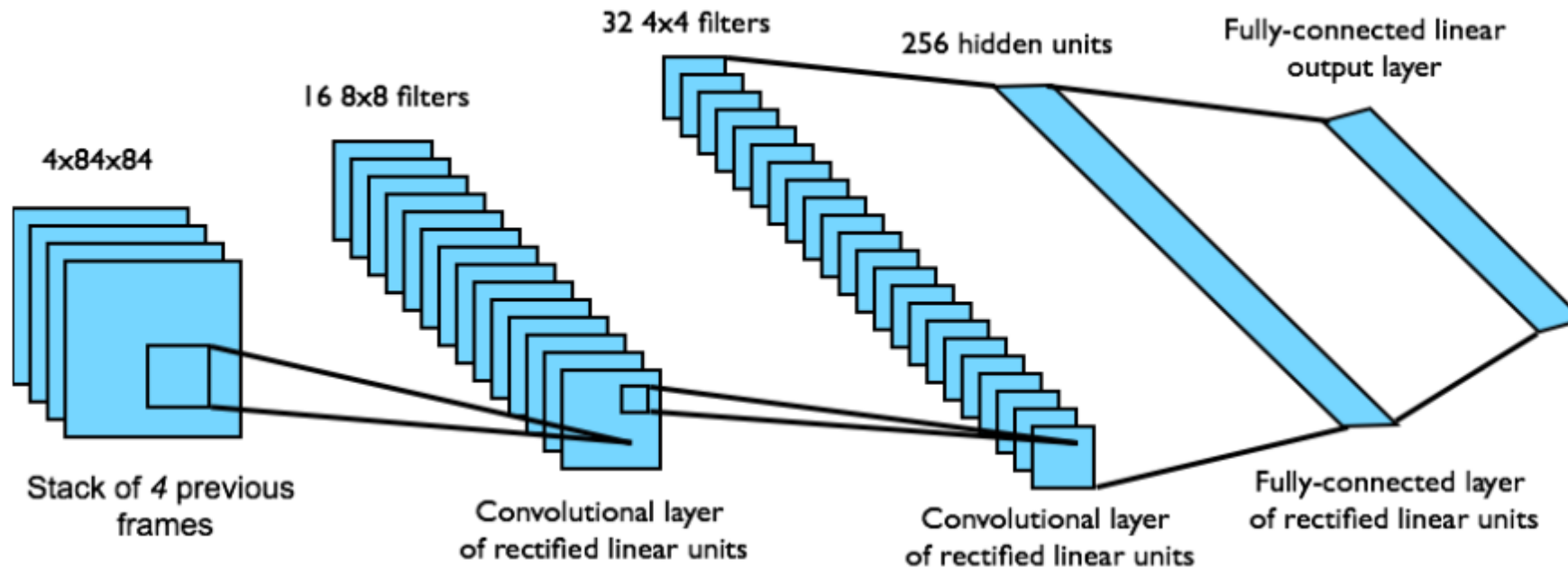


From theory to practice: DQN case



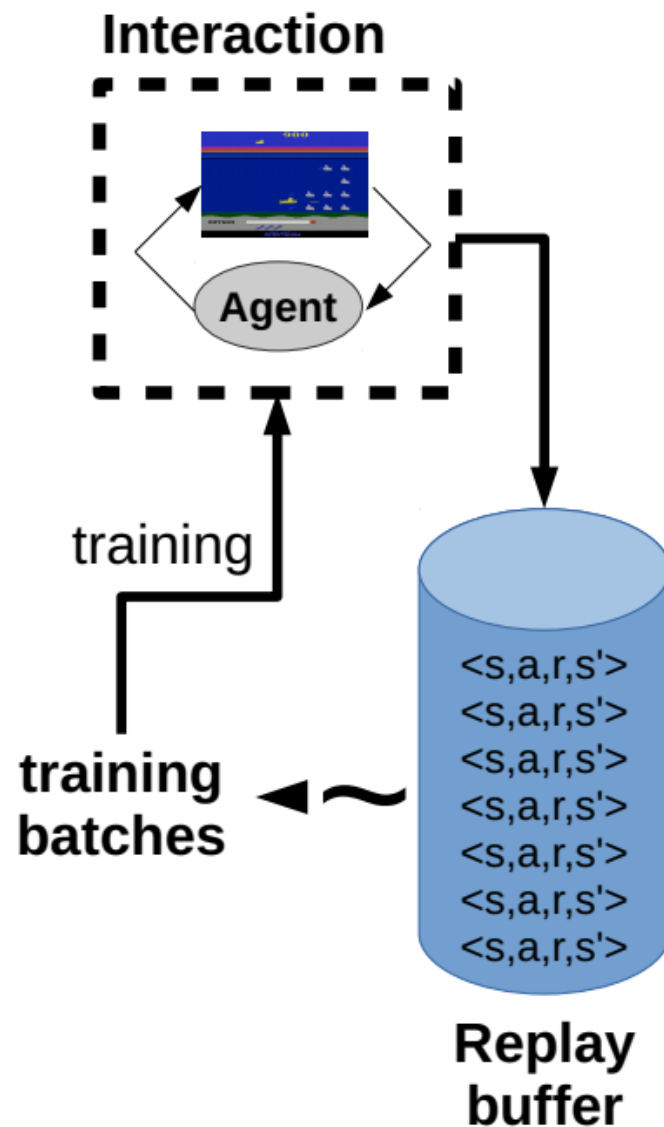
DQN: Atari

- End-to-end learning of values $Q(s, a)$ from pixels s
- Input state s is stack of raw pixels from last **4** frames
- Output is $Q(s, a)$ for **18** joystick/button positions
- Reward is change in score for that step



Experience Replay

- **Idea:** store several past interactions $\langle s, a, r, s' \rangle$
- Train on random subsamples



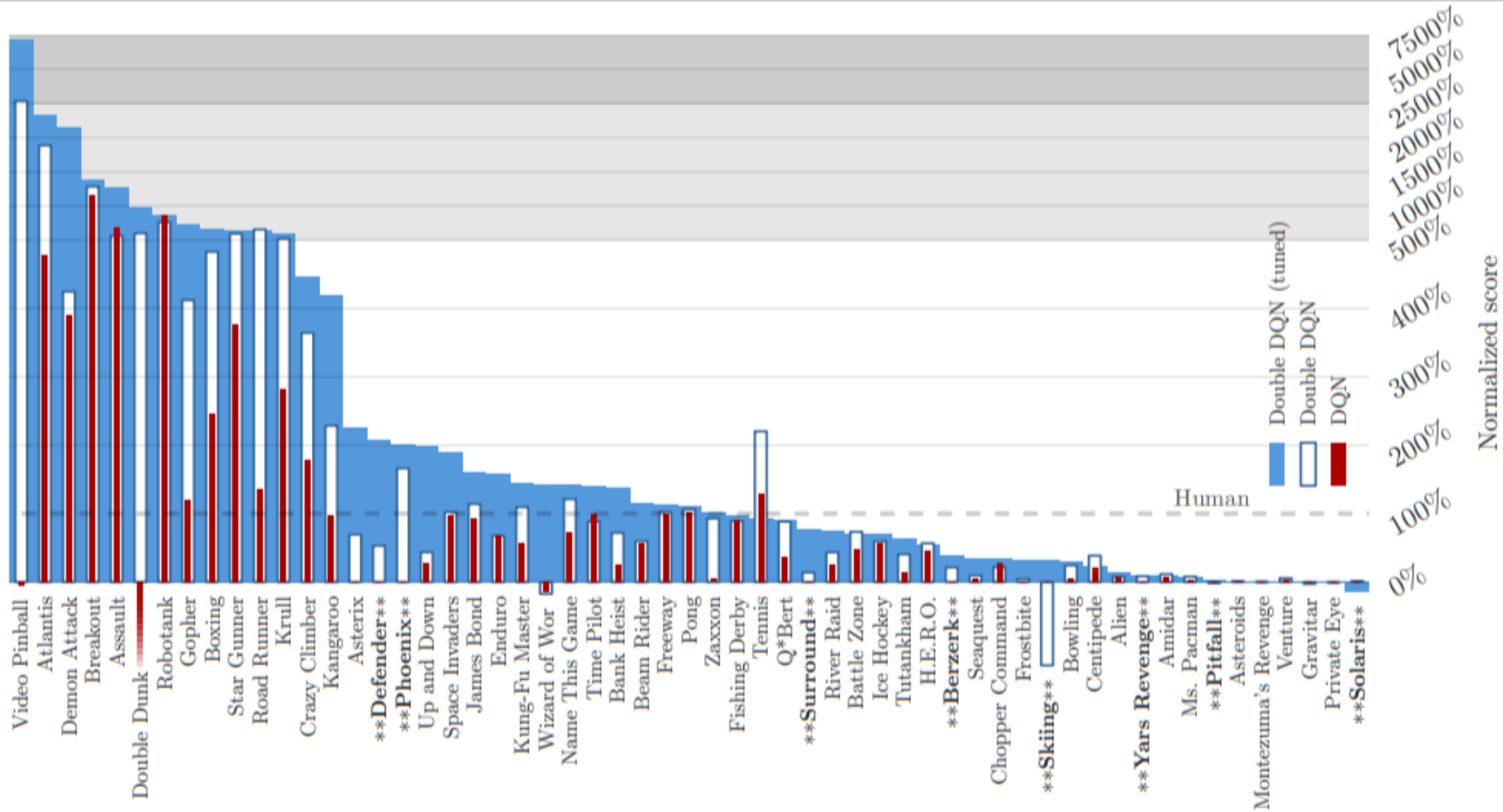
DQN: under the hood

- DQN uses **experience replay** and **fixed Q-targets**:
 - Take action \mathbf{a}_t according to ϵ -greedy policy
 - Store transition $(\mathbf{s}, \mathbf{a}, r, \mathbf{s}')$ in replay memory \mathbf{D}
 - Sample random mini-batch of transitions $(\mathbf{s}, \mathbf{a}, r, \mathbf{s}')$ from \mathbf{D}
 - Compute Q-learning targets w.r.t. old, fixed parameters \mathbf{w}
 - Optimize MSE between Q-network and Q-learning targets

$$\mathcal{L}_i(w_i) = \mathbb{E}_{\mathbf{s}, \mathbf{a}, r, \mathbf{s}' \sim \mathcal{D}_i} \left[\left(r + \gamma \max_{\mathbf{a}'} Q(\mathbf{s}', \mathbf{a}'; w_i^-) - Q(\mathbf{s}, \mathbf{a}; w_i) \right)^2 \right]$$

- Using variant of stochastic gradient descent

DQN results on Atari



DQN: Atari Breakout

- <https://www.youtube.com/watch?v=TmPfTpjtdgg>

RL Literature

- **An Introduction to Reinforcement Learning, Sutton and Barto, 1998**

Available free online!

<http://incompleteideas.net/book/RLbook2020.pdf>

- **Algorithms for Reinforcement Learning, Szepesvari, Morgan and Claypool, 2009**

Available free online!

<https://sites.ualberta.ca/~szepesva/papers/RLAlgsInMDPs.pdf>