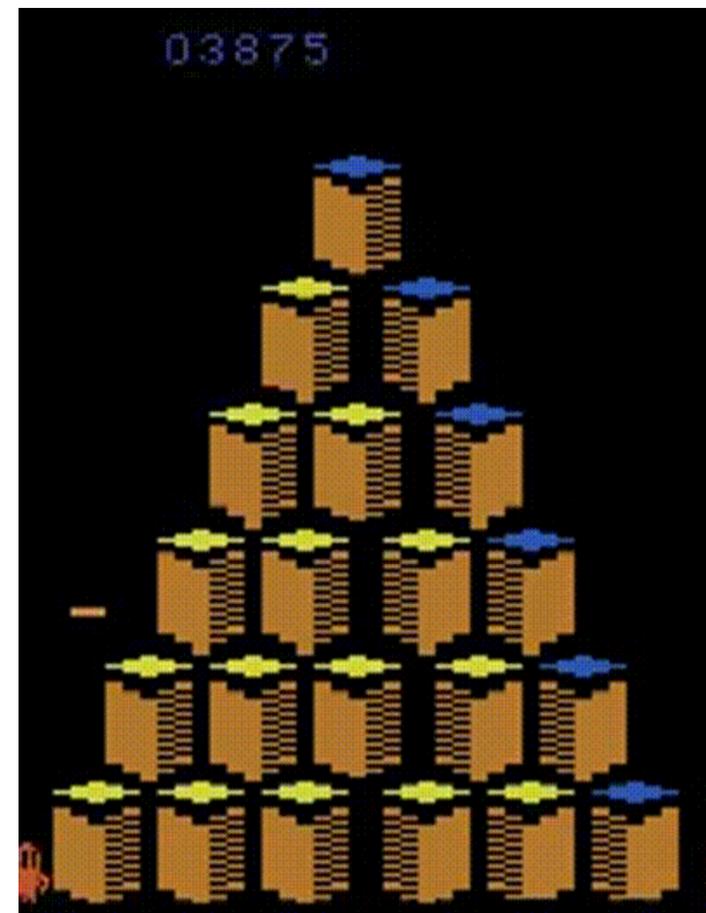
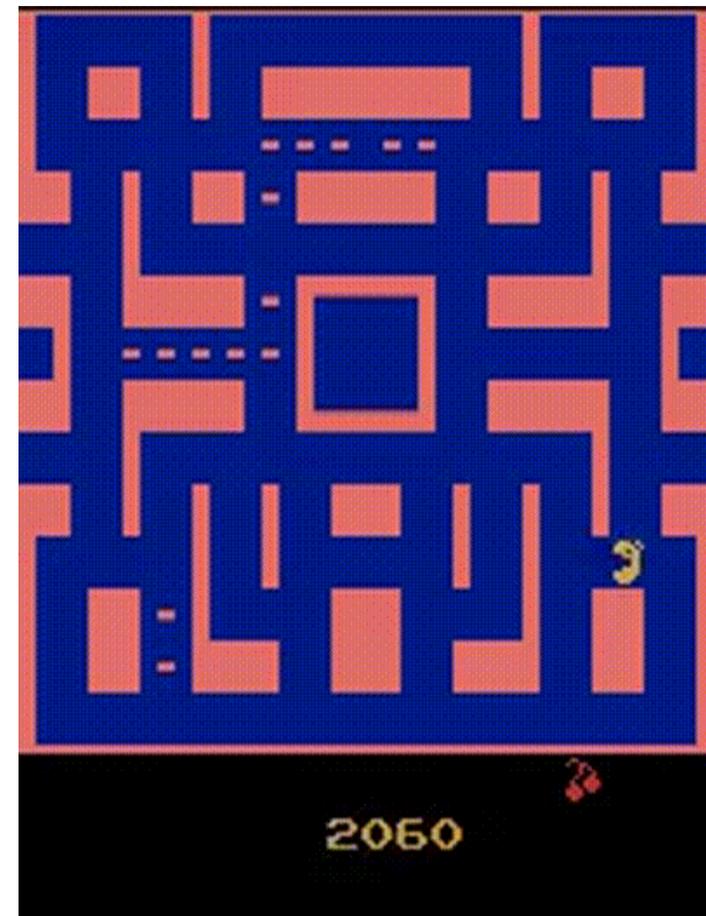


CS 224R Tutorial

Review of Q-Learning

Anikait Singh



Outline of Tutorial

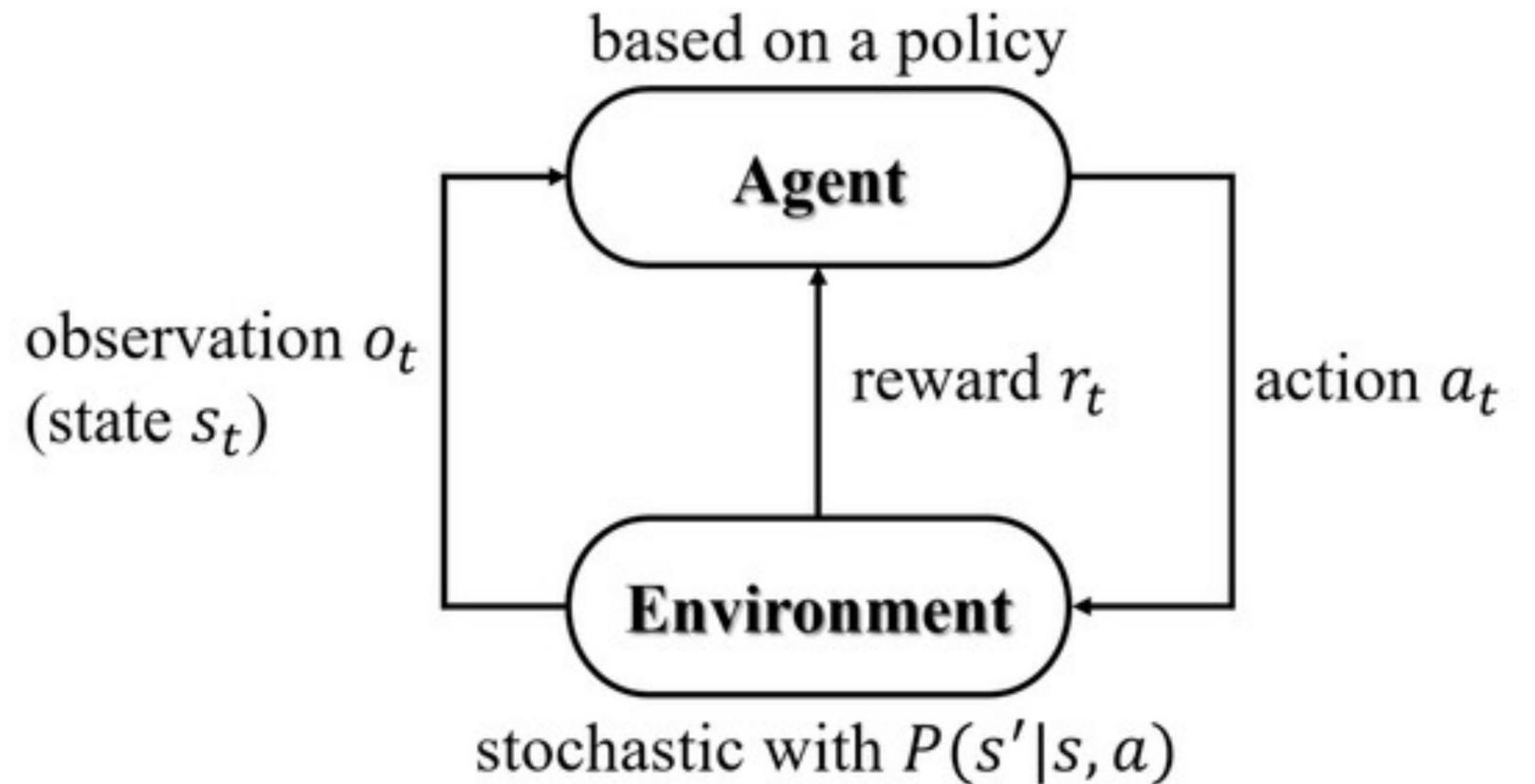
- Review of Markov Decision Processes (MDP)
- Exact MDPs
 - Fitted Q Iteration
- Parametric Q-Learning
 - Bias Variance Tradeoff (TD vs MC)
- Practical Details
 - Replay Buffer, Overestimation, TD Gradients
- Q-Learning/Actor Critic Algorithm Walkthrough (Preview of Homework 2)

Many thanks to Pieter Abbeel, David Silver, Aviral Kumar, and Justin Fu!

MDP Formulation

An MDP is defined by:

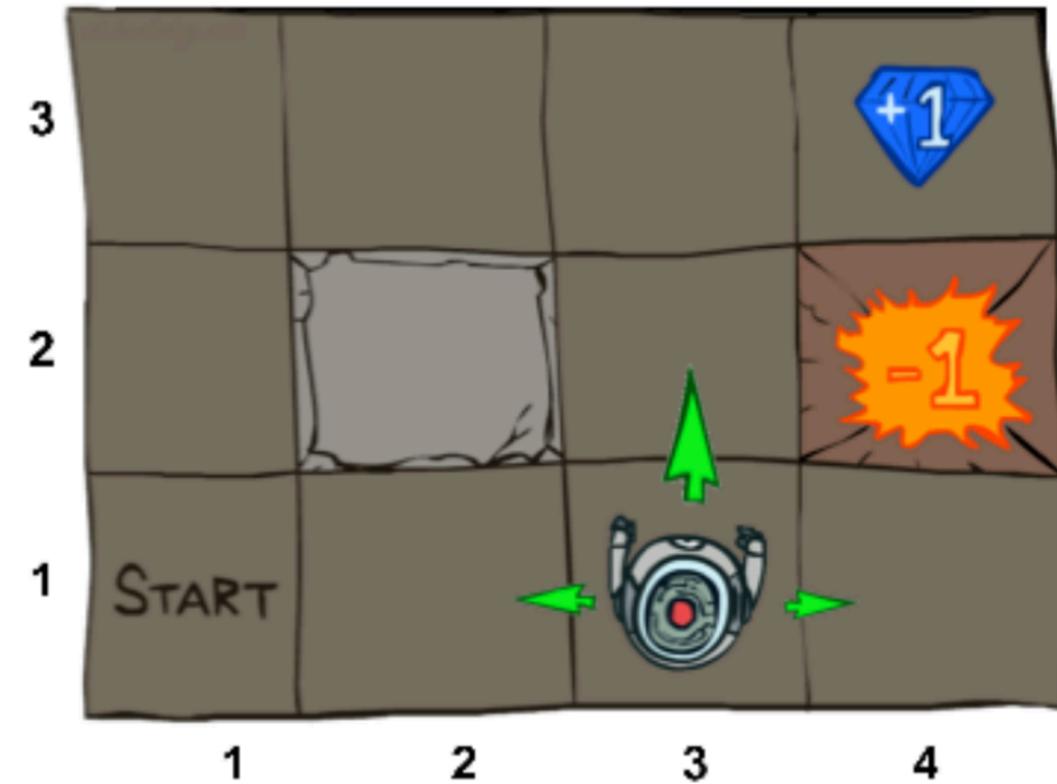
- Set of states s_t
- Set of actions a_t
- Transition function $p(s_{t+1} | s_t, a_t)$
- Reward Function $r(s_t, a_t)$
- Start state s_1
- Discount Factor γ
- Horizon T



MDP Formulation

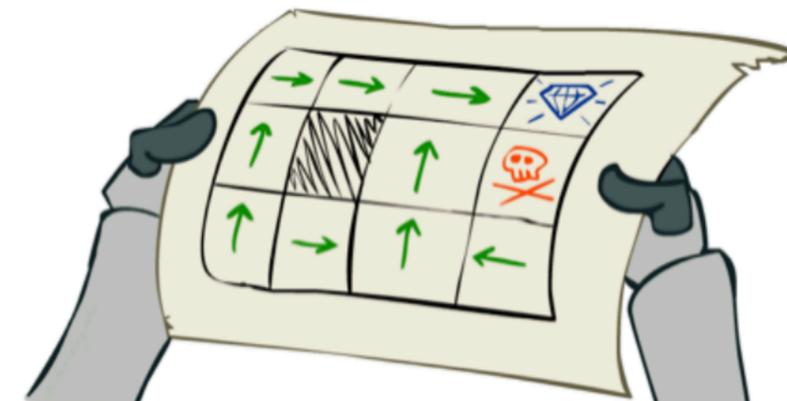
An MDP is defined by:

- Set of states s_t
- Set of actions a_t
- Transition function $p(s_{t+1} | s_t, a_t)$
- Reward Function $r(s_t, a_t)$
- Start state s_1
- Discount Factor γ
- Horizon T



Goal: $\max_{\pi} \mathbb{E} \left[\sum_{t=1}^T \gamma^t r(s_t, a_t) \mid \pi \right]$

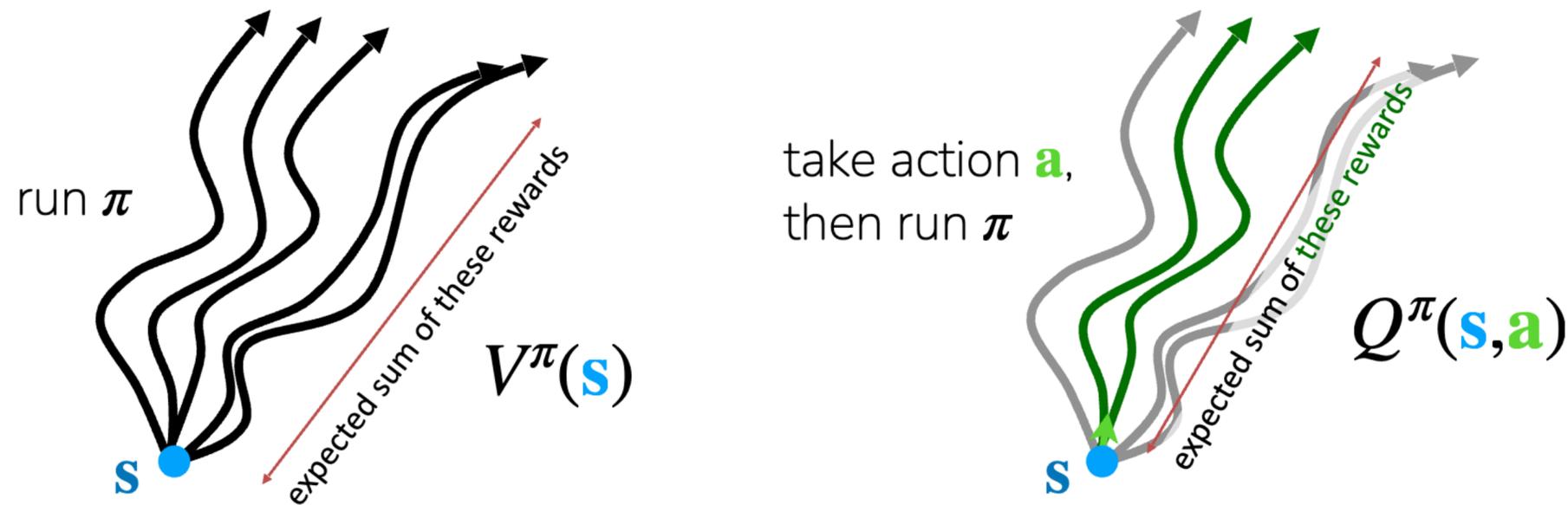
π :



Revisiting Some Useful Objects

value function $V^\pi(\mathbf{s})$ - future expected rewards starting at \mathbf{s} and following π

Q-function $Q^\pi(\mathbf{s}, \mathbf{a})$ - future expected rewards starting at \mathbf{s} , taking \mathbf{a} , then following π



Useful relation: $V^\pi(\mathbf{s}) = \mathbb{E}_{\mathbf{a} \sim \pi(\cdot | \mathbf{s})} [Q^\pi(\mathbf{s}, \mathbf{a})]$

advantage $A^\pi(\mathbf{s}, \mathbf{a})$ - how much better it is to take \mathbf{a} than to follow policy π at state \mathbf{s}

$$A^\pi(\mathbf{s}, \mathbf{a}) = Q^\pi(\mathbf{s}, \mathbf{a}) - V^\pi(\mathbf{s})$$

Tabular Learning - Fitted Q Iteration

Q-Function $Q^\pi(\mathbf{s}, \mathbf{a})$ - future expected rewards starting at state \mathbf{s} , taking action \mathbf{a} , then following π

Bellman Equation

$$Q^*(s, a) = \sum_{s_{t+1}} p(s_{t+1} | s_t, a_t) \left(r(s_t, a_t) + \gamma \max_{a_t} Q^*(s_{t+1}, a_{t+1}) \right)$$

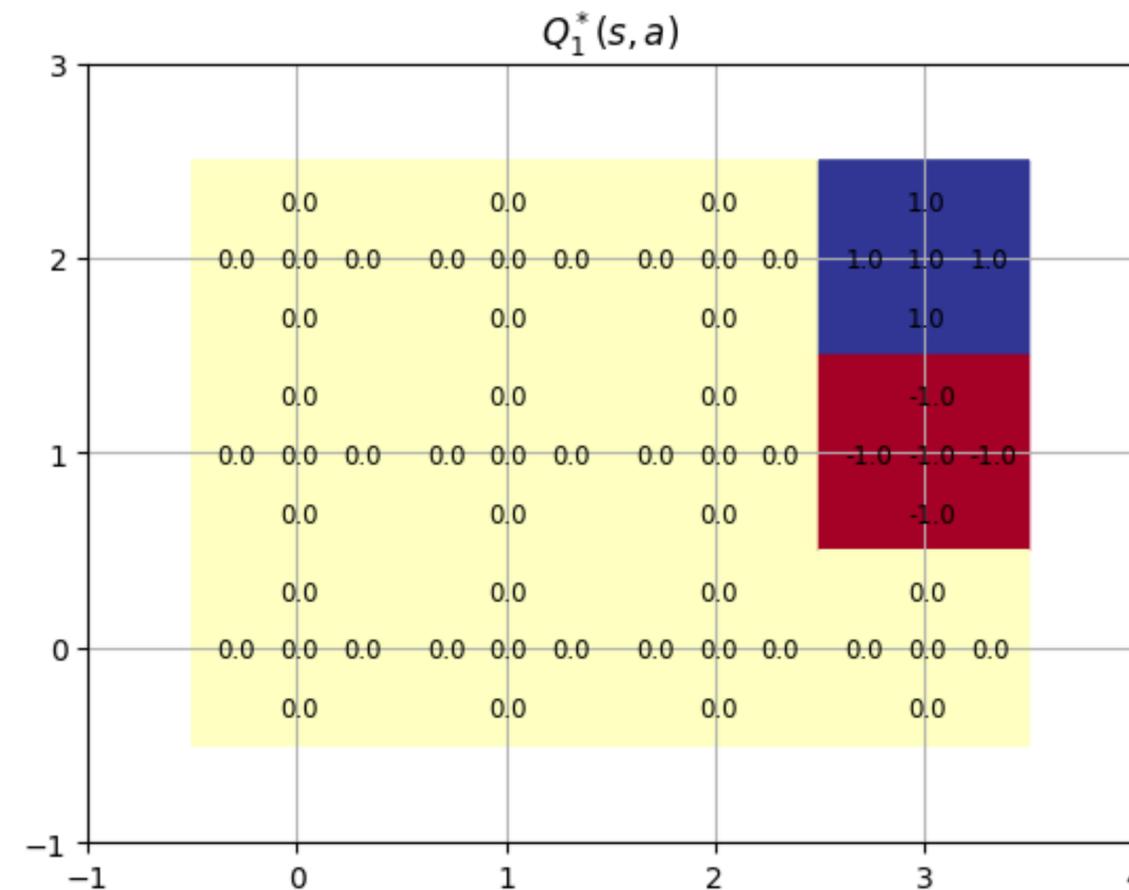
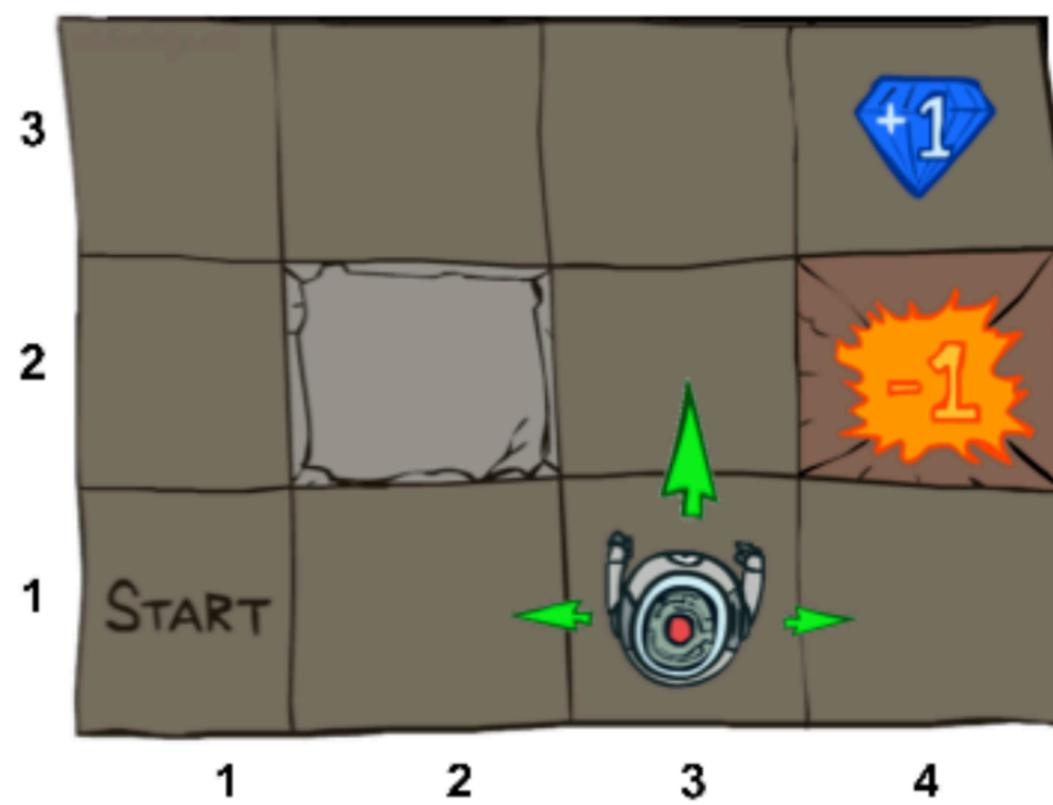
Q-Value Iteration

$$Q_{k+1}^*(s, a) = \sum_{s_{t+1}} p(s_{t+1} | s_t, a_t) \left(r(s_t, a_t) + \gamma \max_{a_t} Q_k^*(s_{t+1}, a_{t+1}) \right)$$

Fitted Q Iteration with the MDP

Discount = 0.9

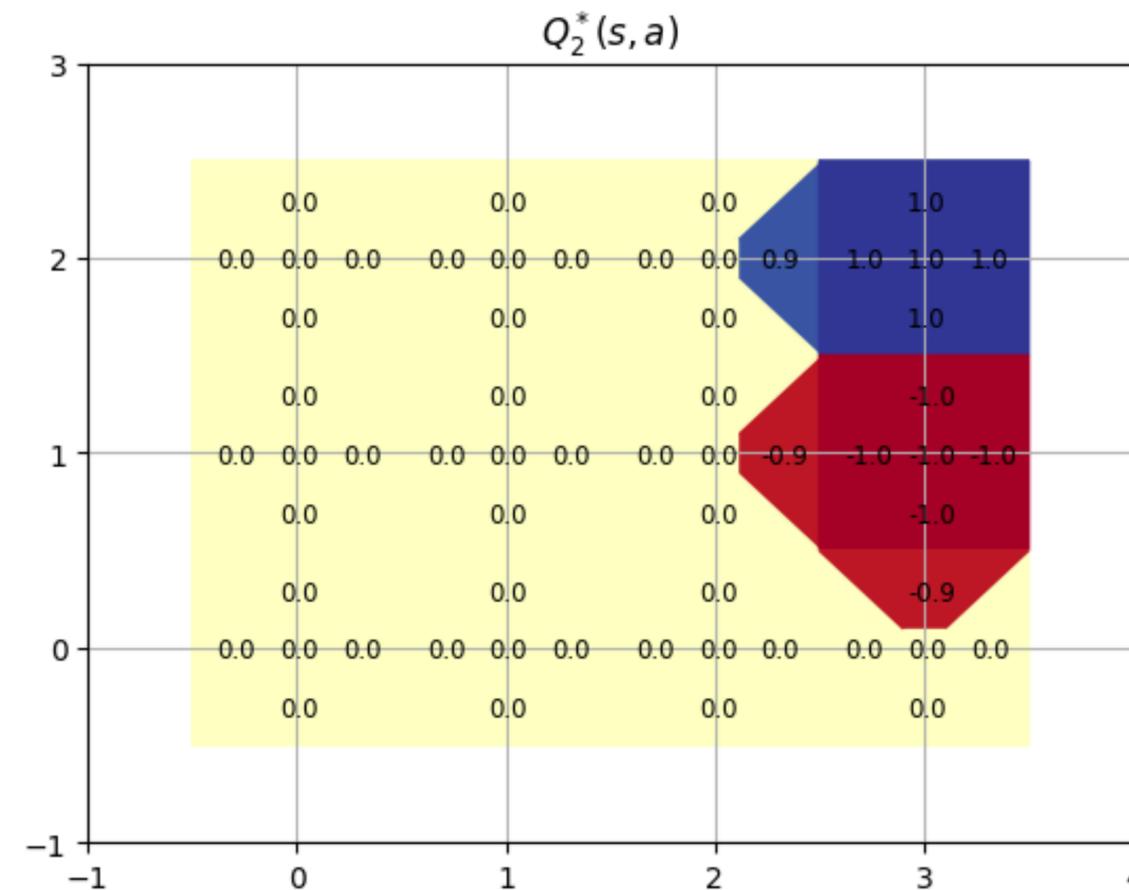
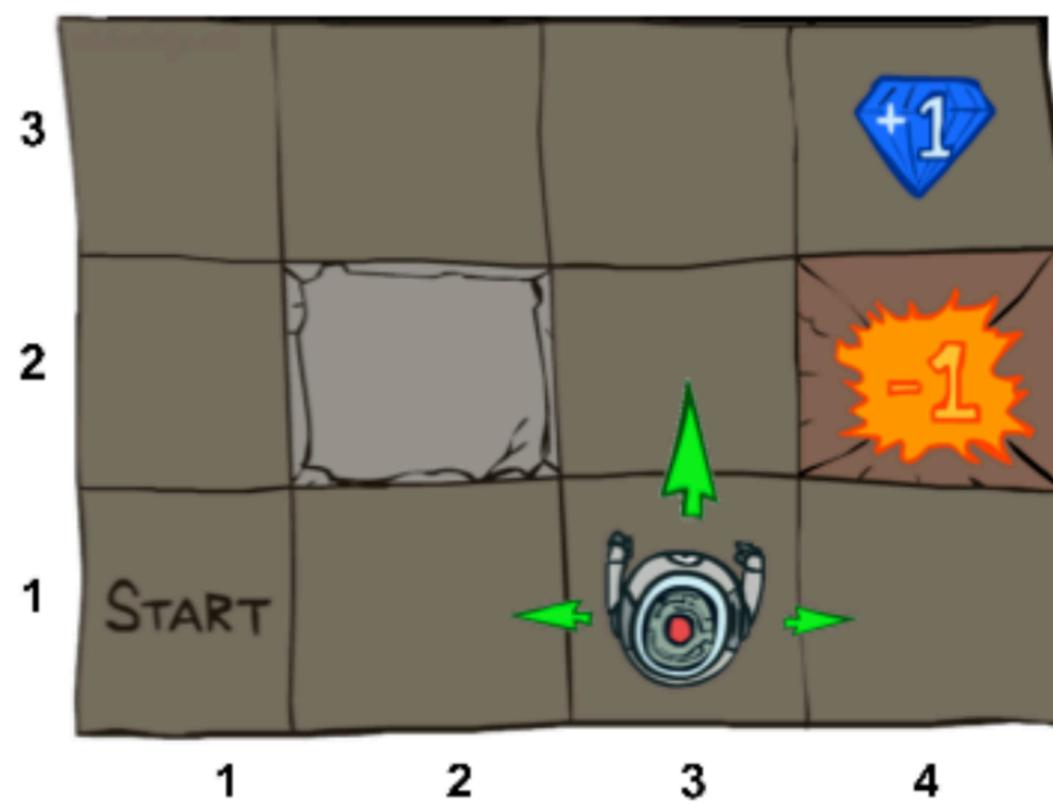
$$Q_{k+1}^*(s, a) = \sum_{s_{t+1}} p(s_{t+1} | s_t, a_t) \left(r(s_t, a_t) + \gamma \max_{a_t} Q_k^*(s_{t+1}, a_{t+1}) \right)$$



Fitted Q Iteration with the MDP

Discount = 0.9

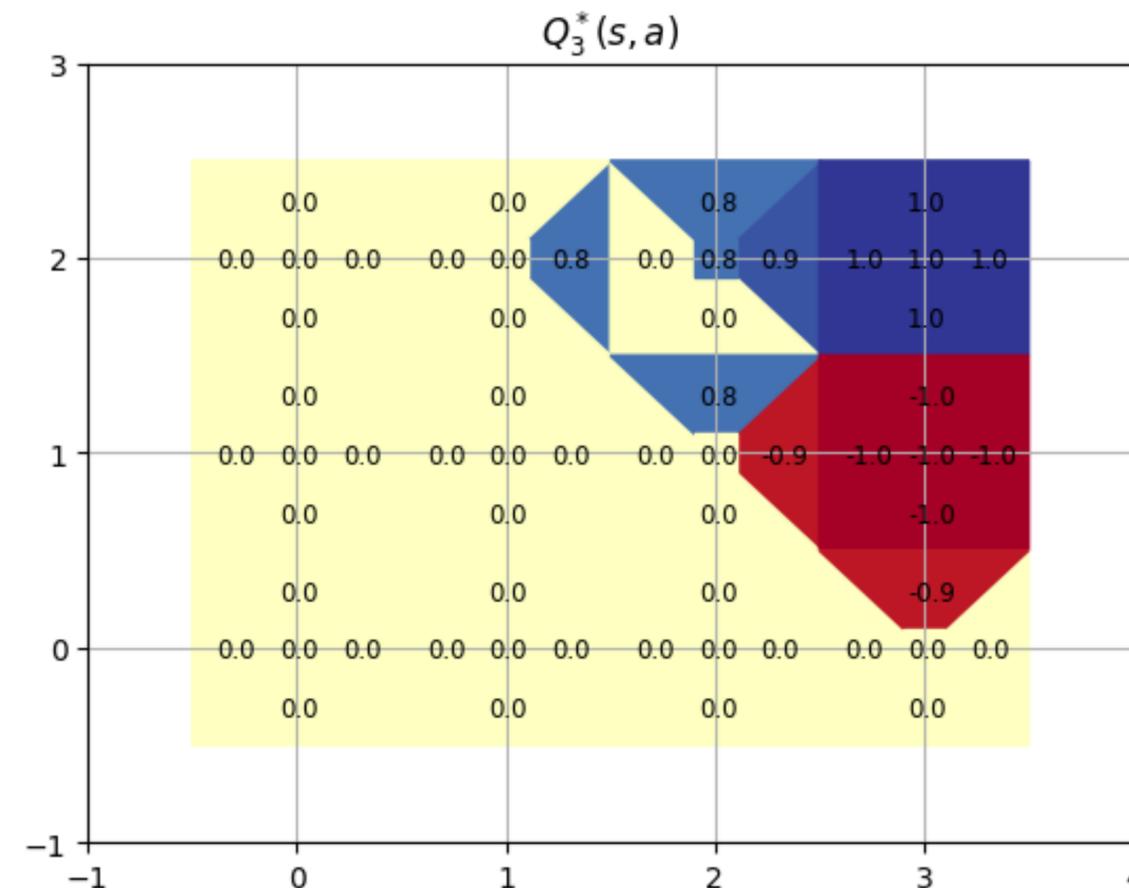
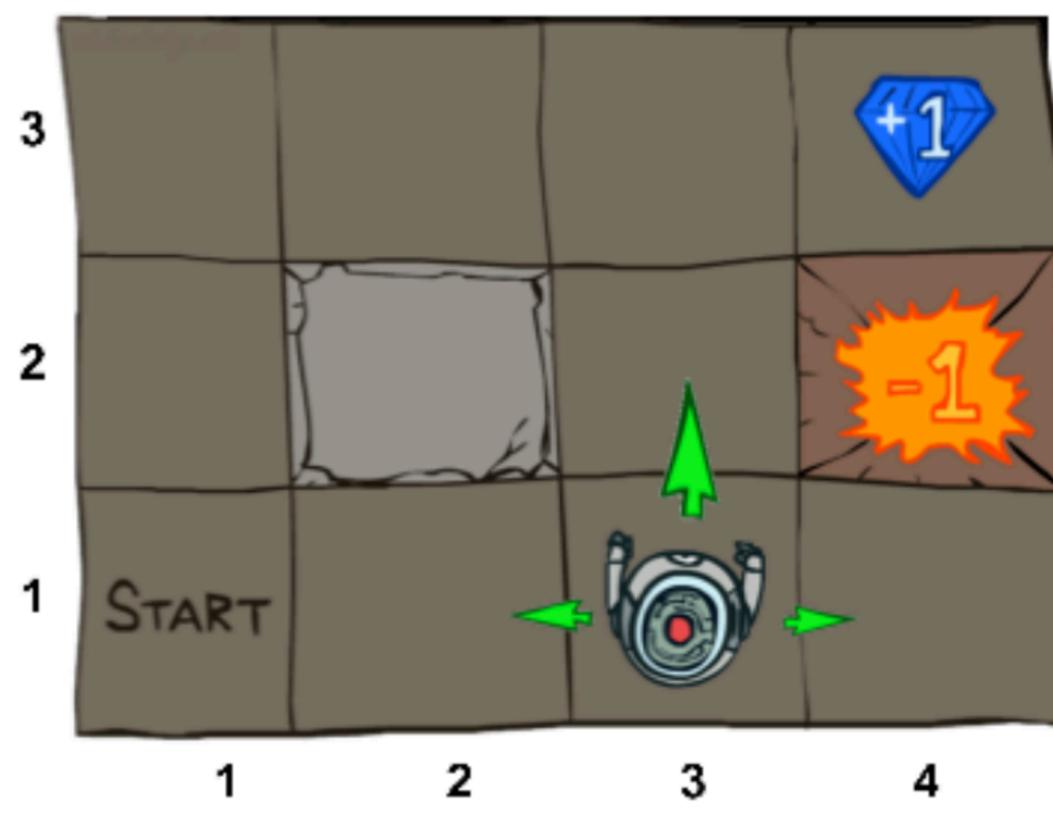
$$Q_{k+1}^*(s, a) = \sum_{s_{t+1}} p(s_{t+1} | s_t, a_t) \left(r(s_t, a_t) + \gamma \max_{a_t} Q_k^*(s_{t+1}, a_{t+1}) \right)$$



Fitted Q Iteration with the MDP

Discount = 0.9

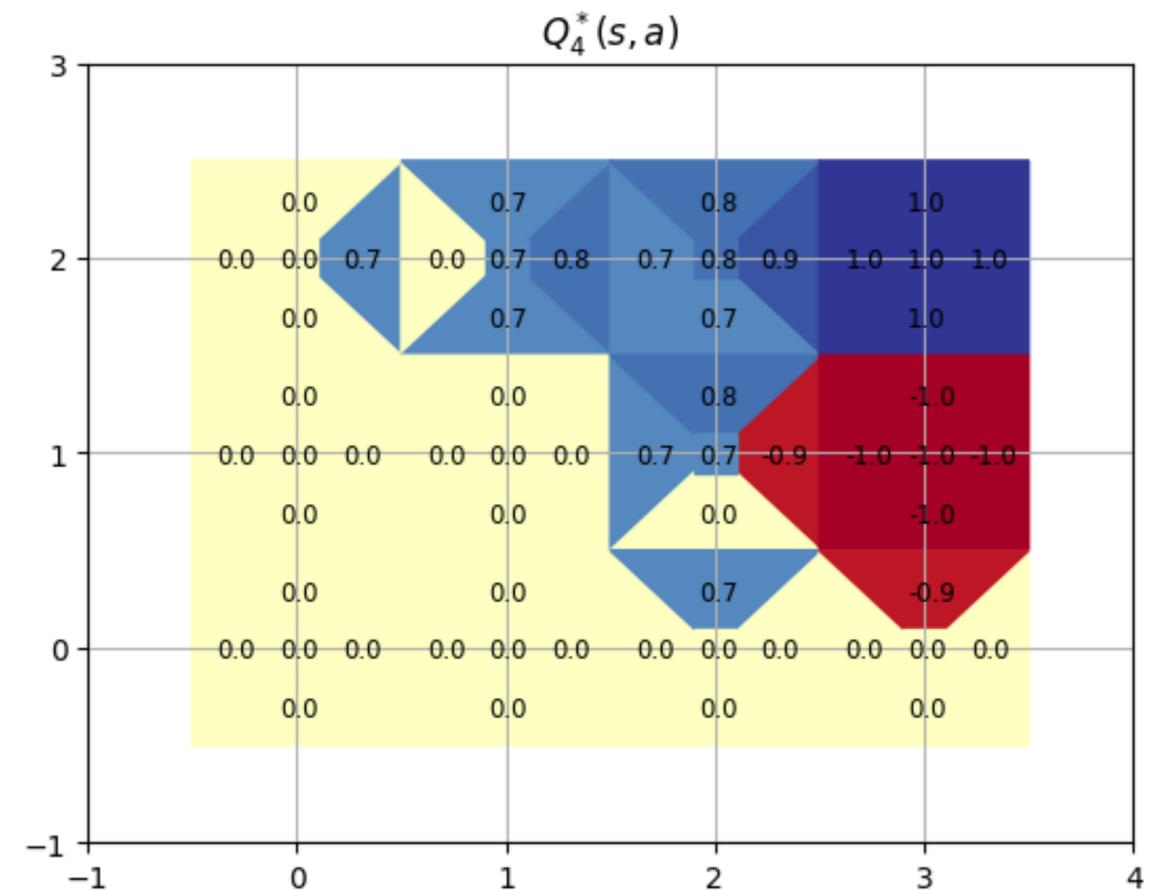
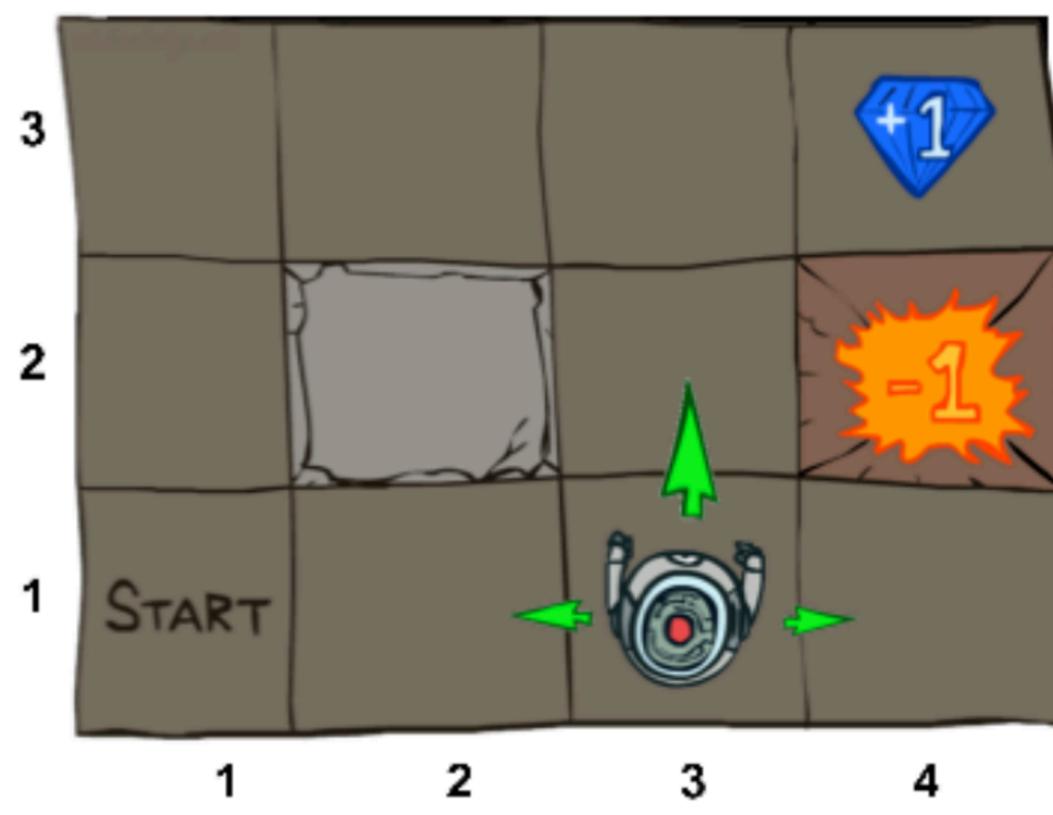
$$Q_{k+1}^*(s, a) = \sum_{s_{t+1}} p(s_{t+1} | s_t, a_t) \left(r(s_t, a_t) + \gamma \max_{a_t} Q_k^*(s_{t+1}, a_{t+1}) \right)$$



Fitted Q Iteration with the MDP

Discount = 0.9

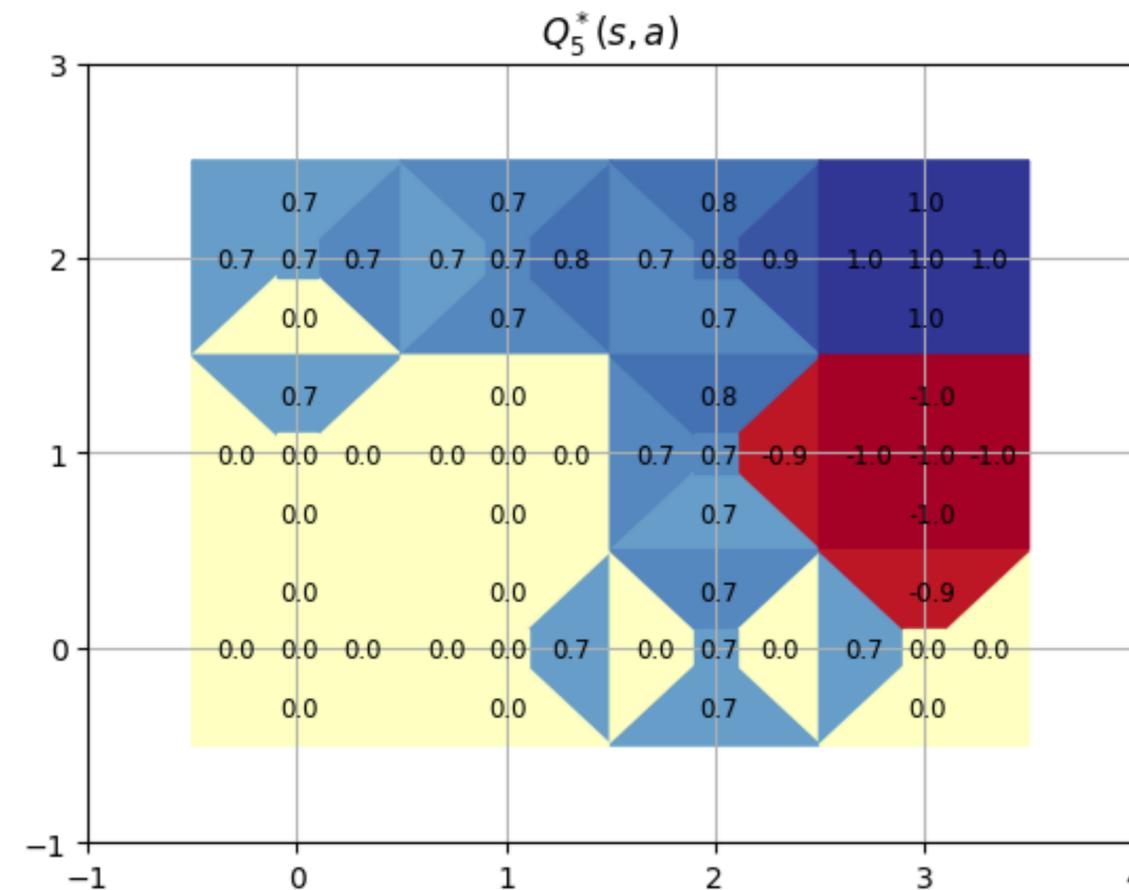
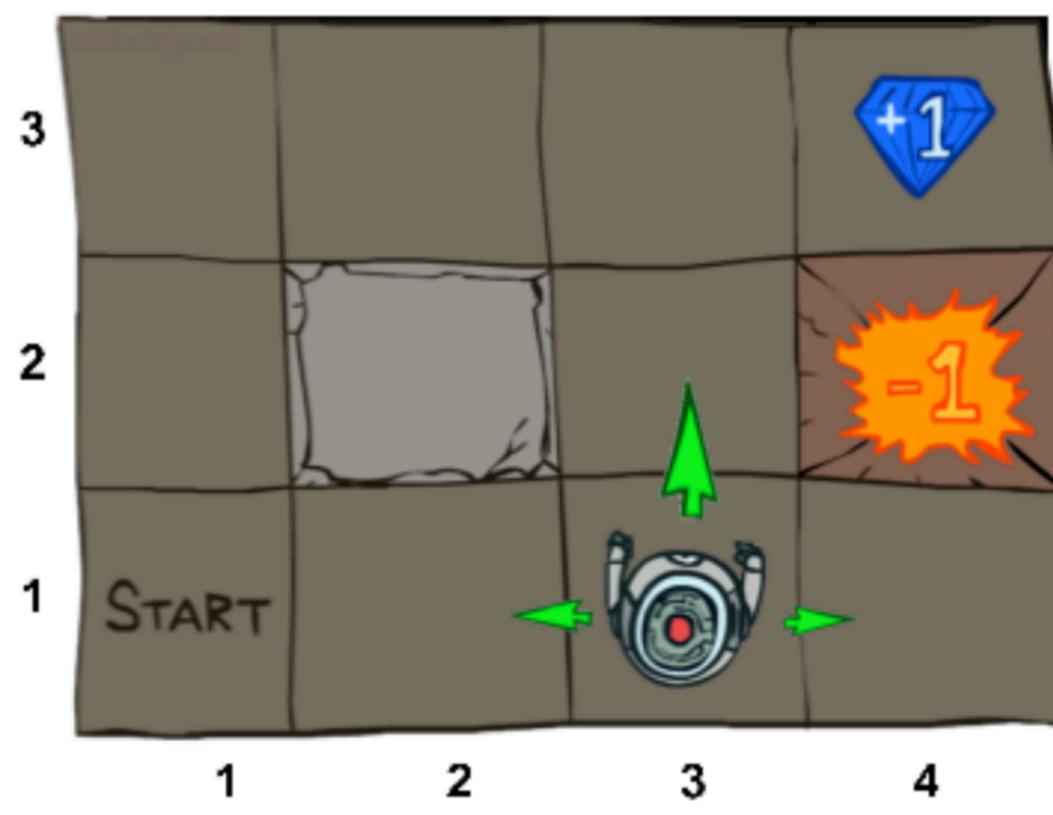
$$Q_{k+1}^*(s, a) = \sum_{s_{t+1}} p(s_{t+1} | s_t, a_t) \left(r(s_t, a_t) + \gamma \max_{a_t} Q_k^*(s_{t+1}, a_{t+1}) \right)$$



Fitted Q Iteration with the MDP

Discount = 0.9

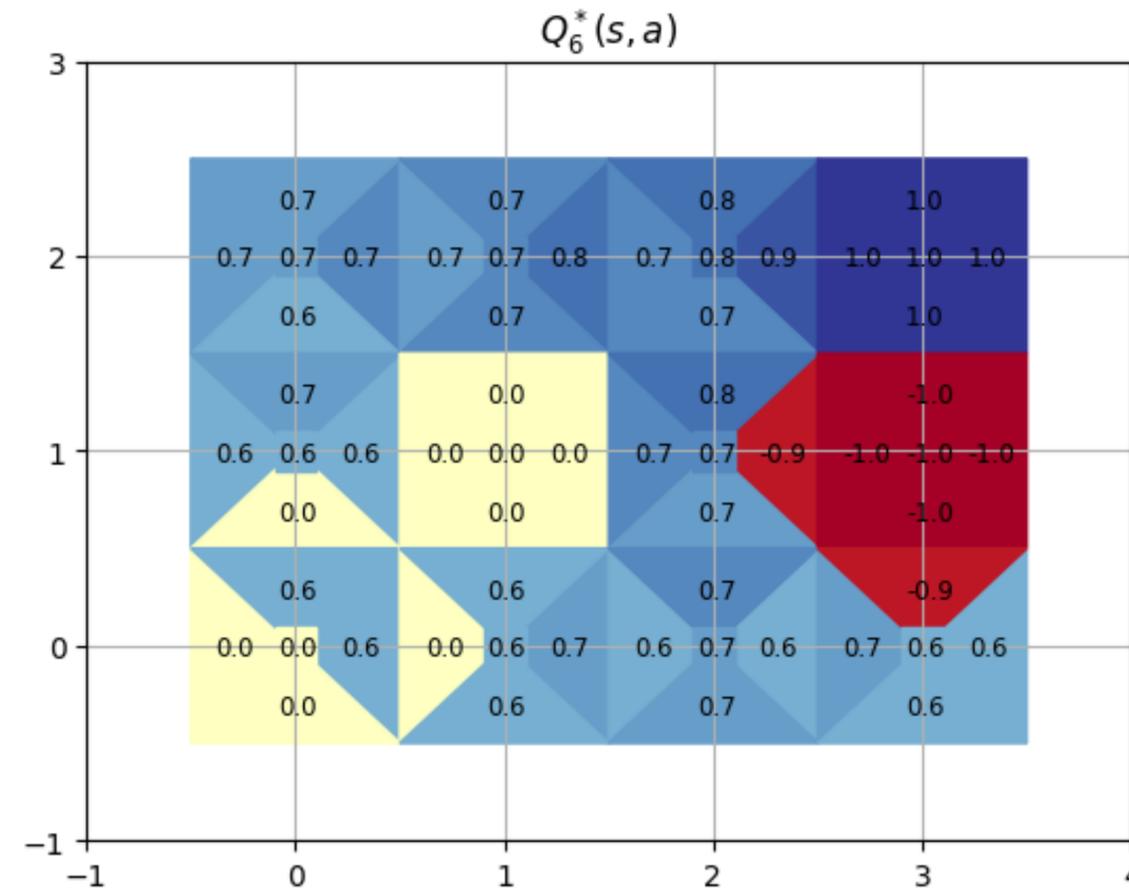
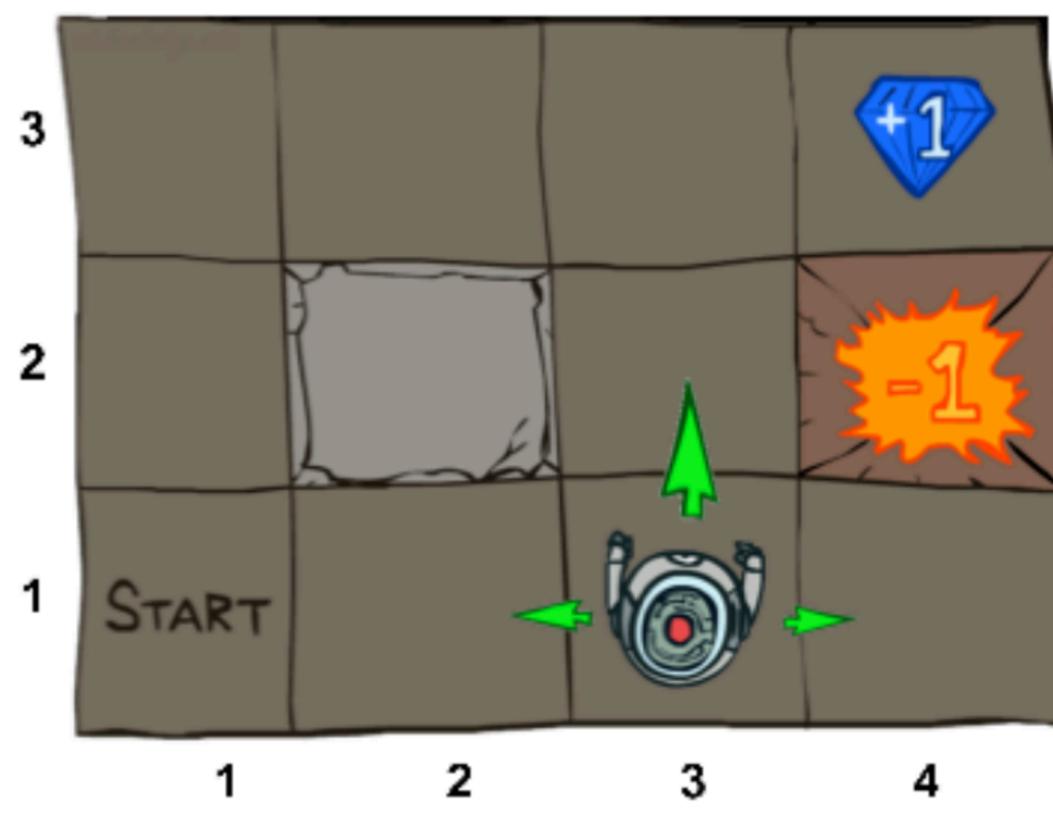
$$Q_{k+1}^*(s, a) = \sum_{s_{t+1}} p(s_{t+1} | s_t, a_t) \left(r(s_t, a_t) + \gamma \max_{a_t} Q_k^*(s_{t+1}, a_{t+1}) \right)$$



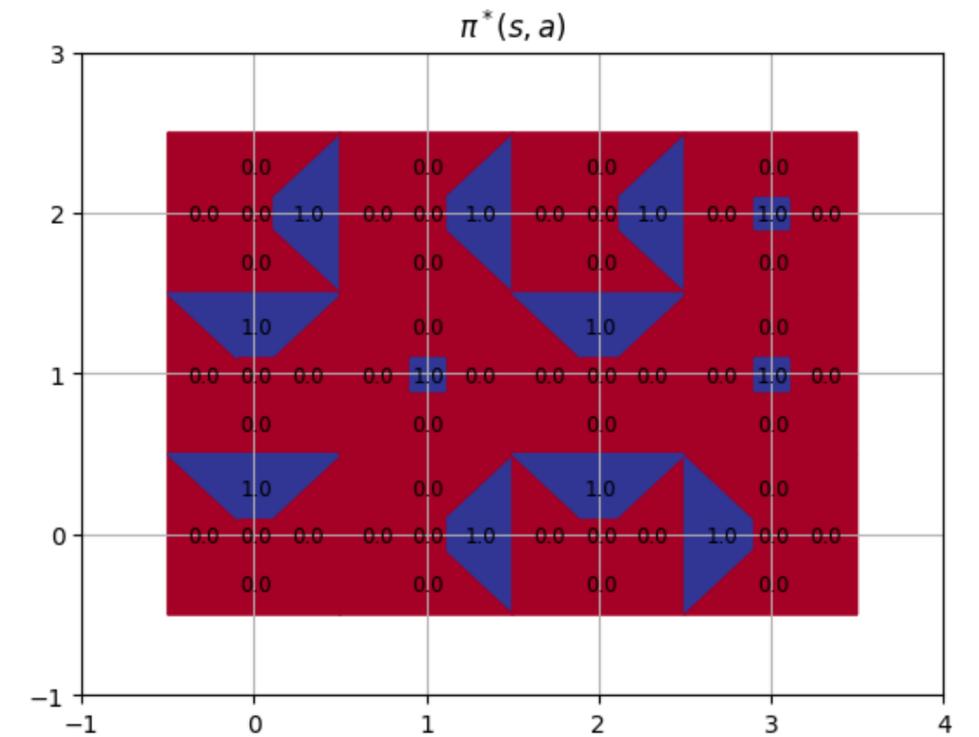
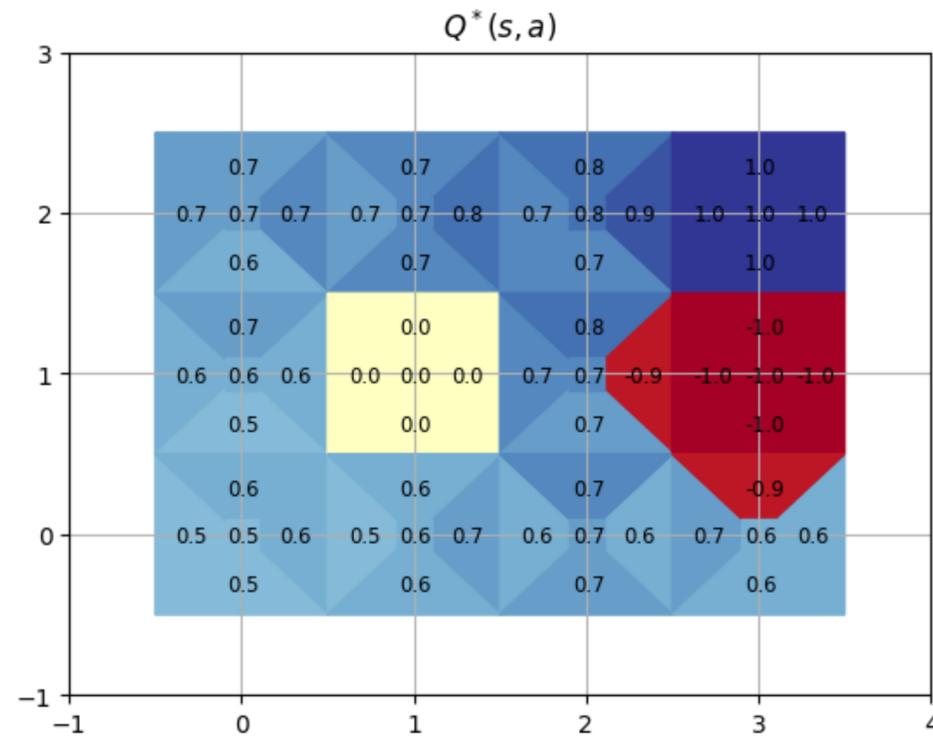
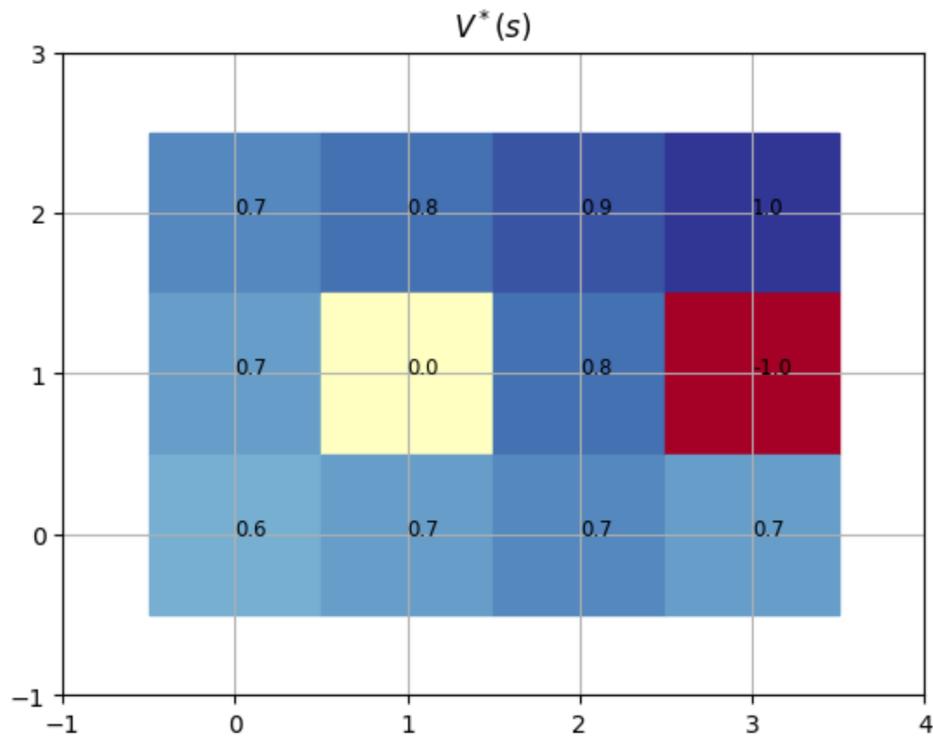
Fitted Q Iteration with the MDP

Discount = 0.9

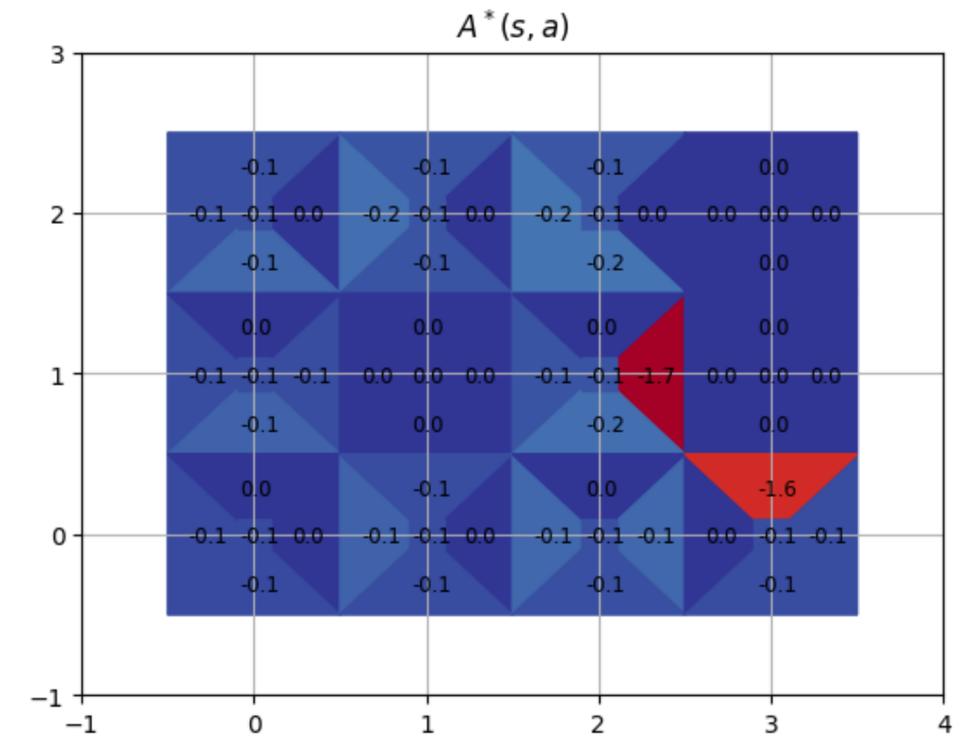
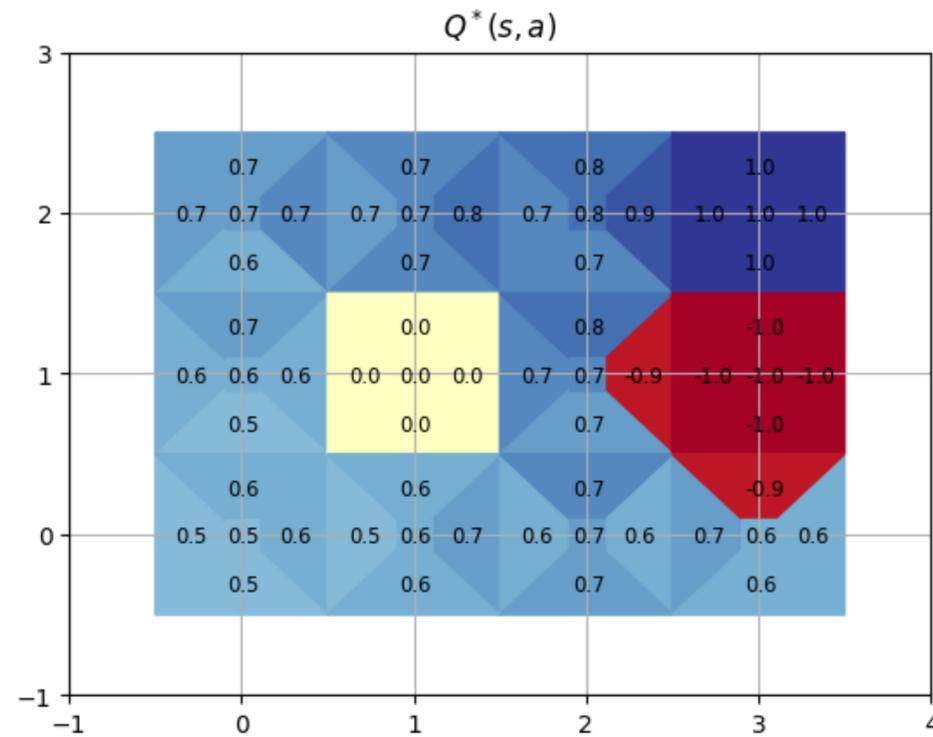
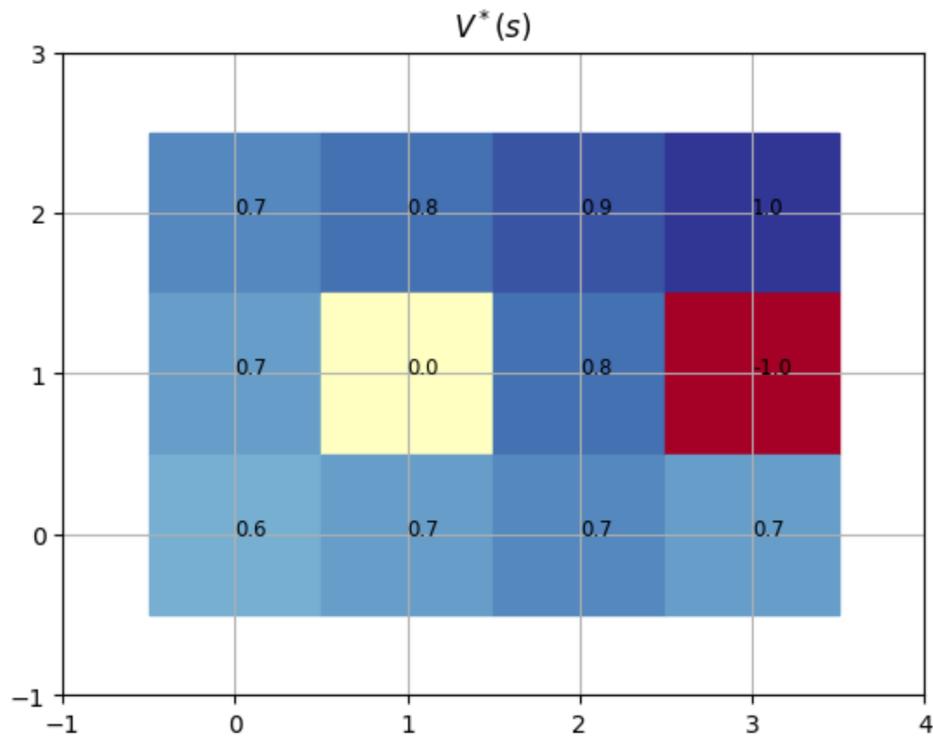
$$Q_{k+1}^*(s, a) = \sum_{s_{t+1}} p(s_{t+1} | s_t, a_t) \left(r(s_t, a_t) + \gamma \max_{a_t} Q_k^*(s_{t+1}, a_{t+1}) \right)$$



Optimal Value, Q-Value, Policy



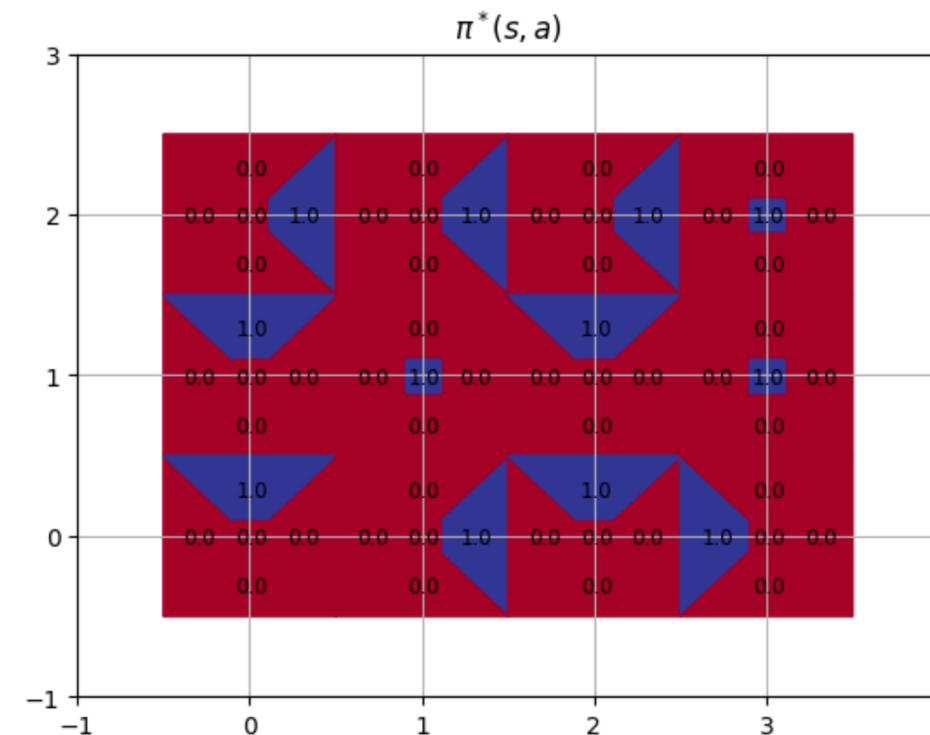
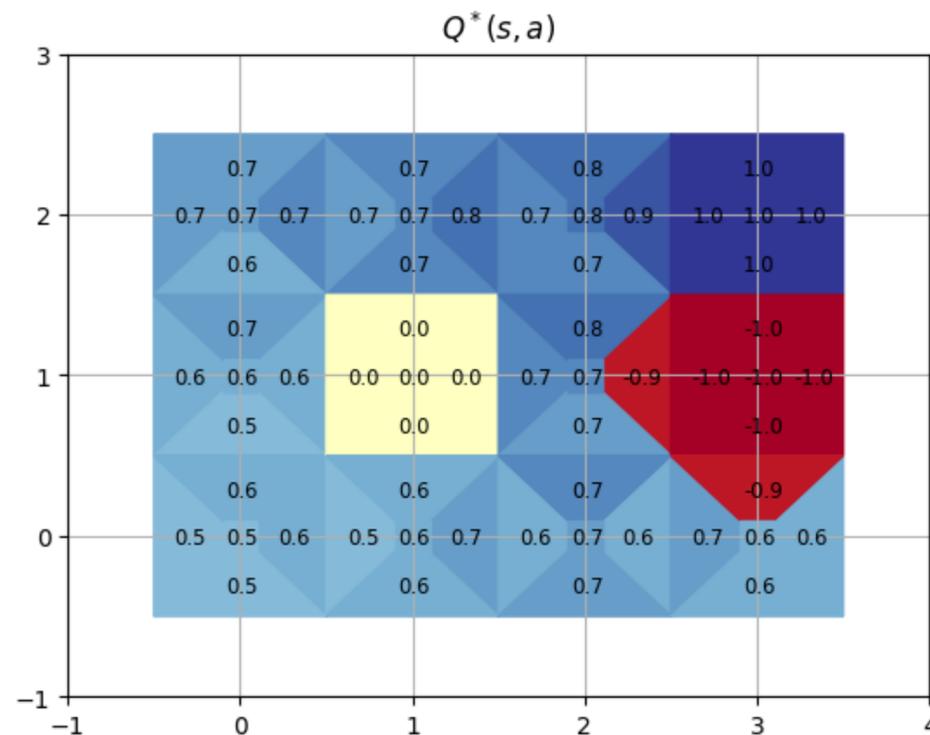
Advantage (Optimal Policy)



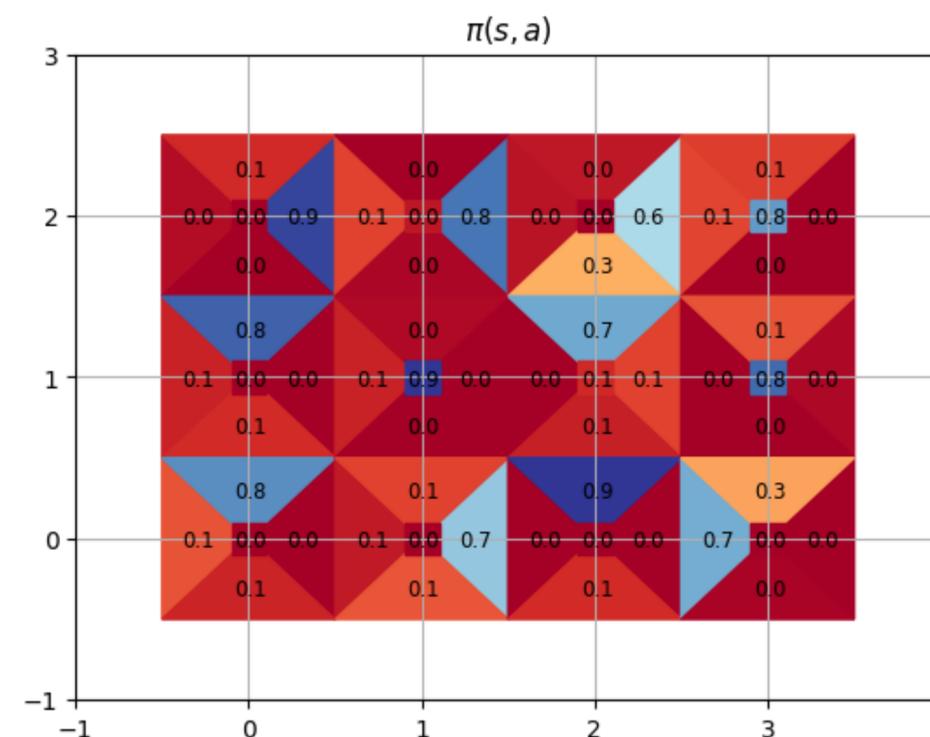
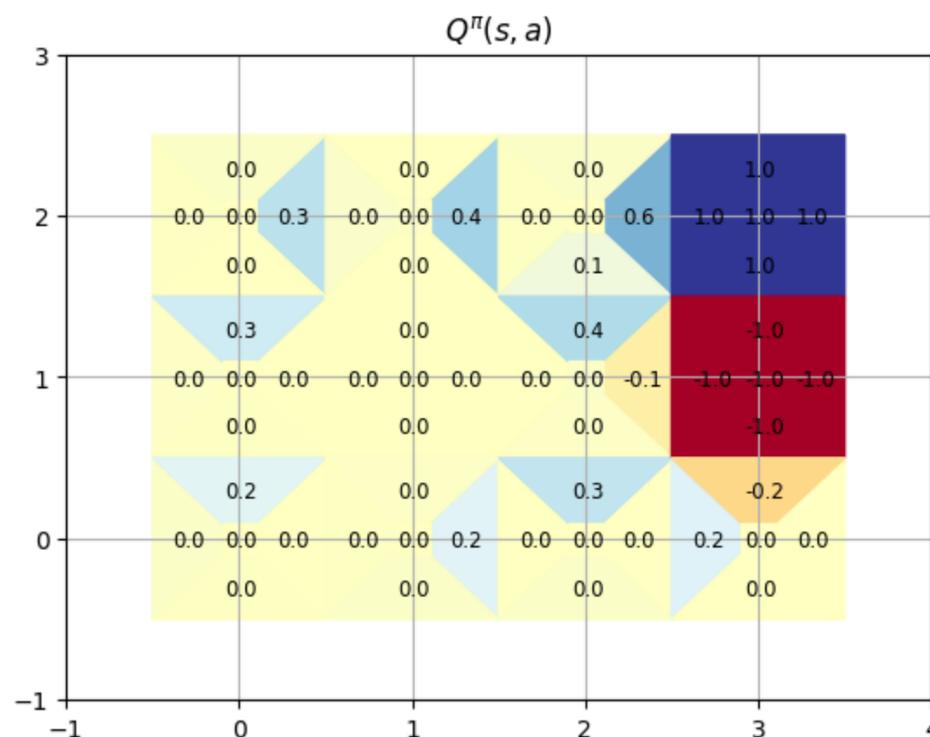
$A^\pi(\mathbf{s}_t, \mathbf{a}_t) = Q^\pi(\mathbf{s}_t, \mathbf{a}_t) - V^\pi(\mathbf{s}_t)$: how much better \mathbf{a}_t is

Q^* vs Q^π

Optimal Policy
(Deterministic for MDP)



Suboptimal Policy



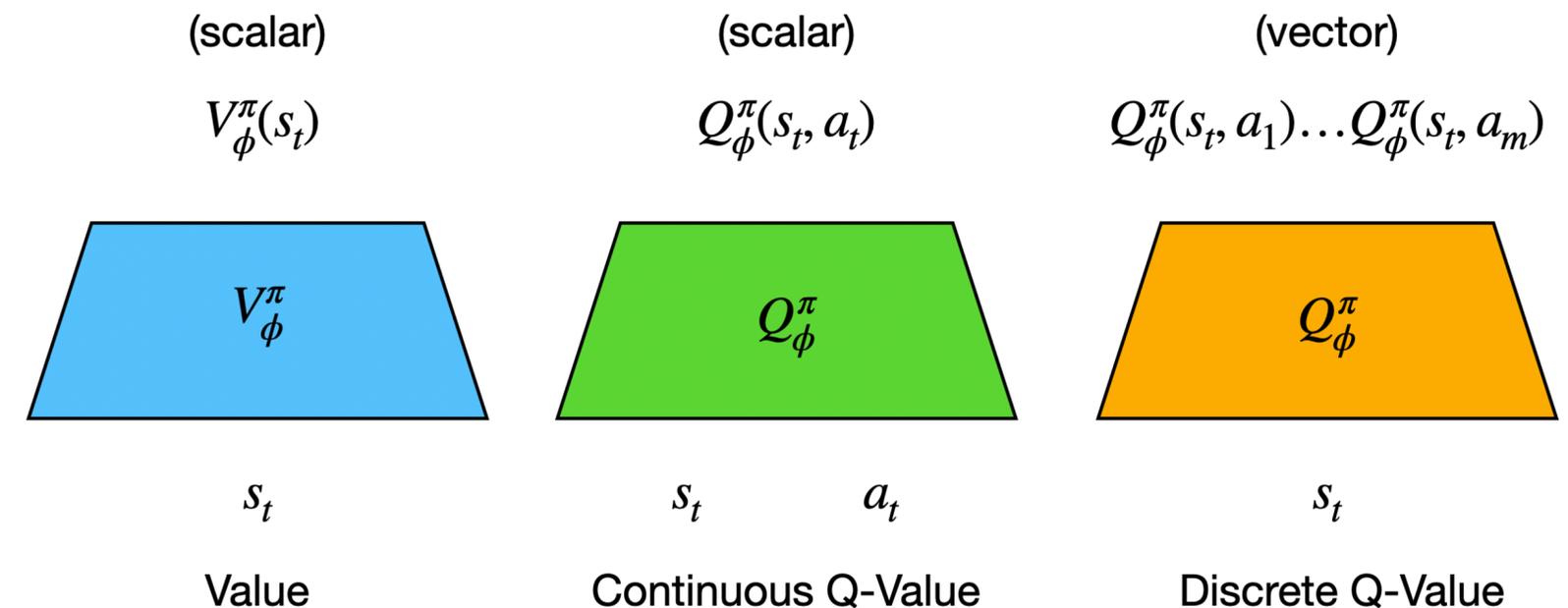
To play more with the Tabular Environment, checkout the [colab](#)

Pause for Questions.

Limitations of Tabular Environment

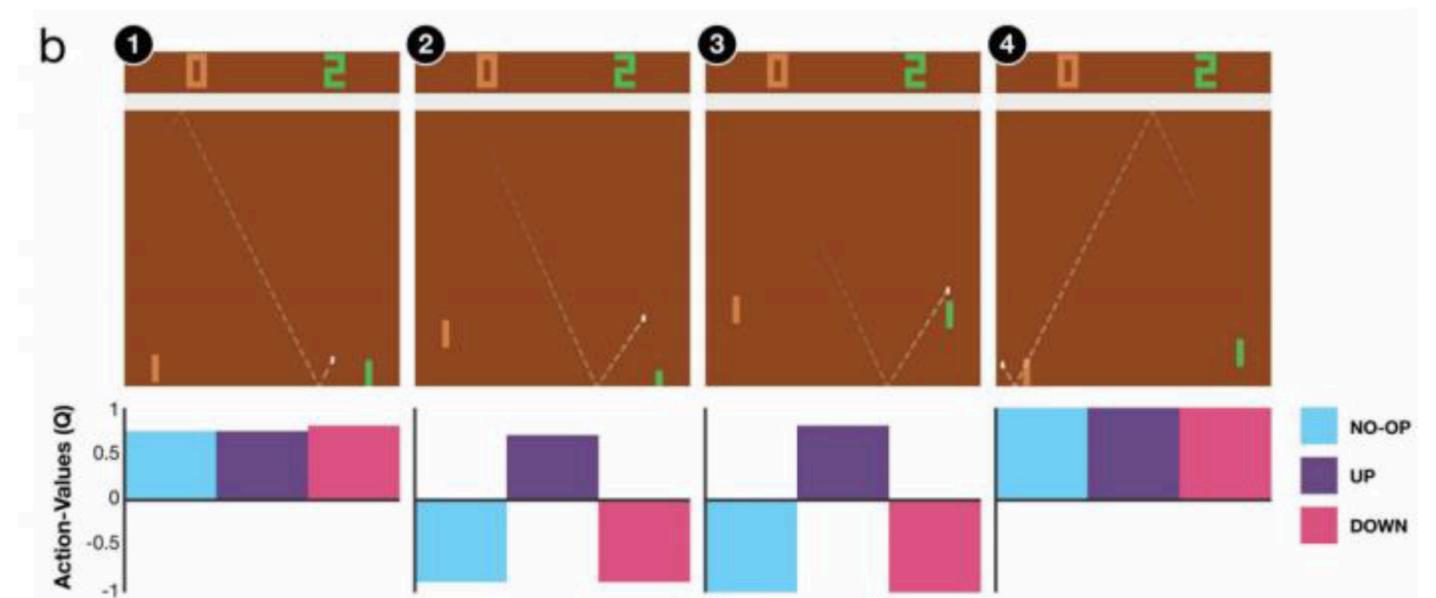
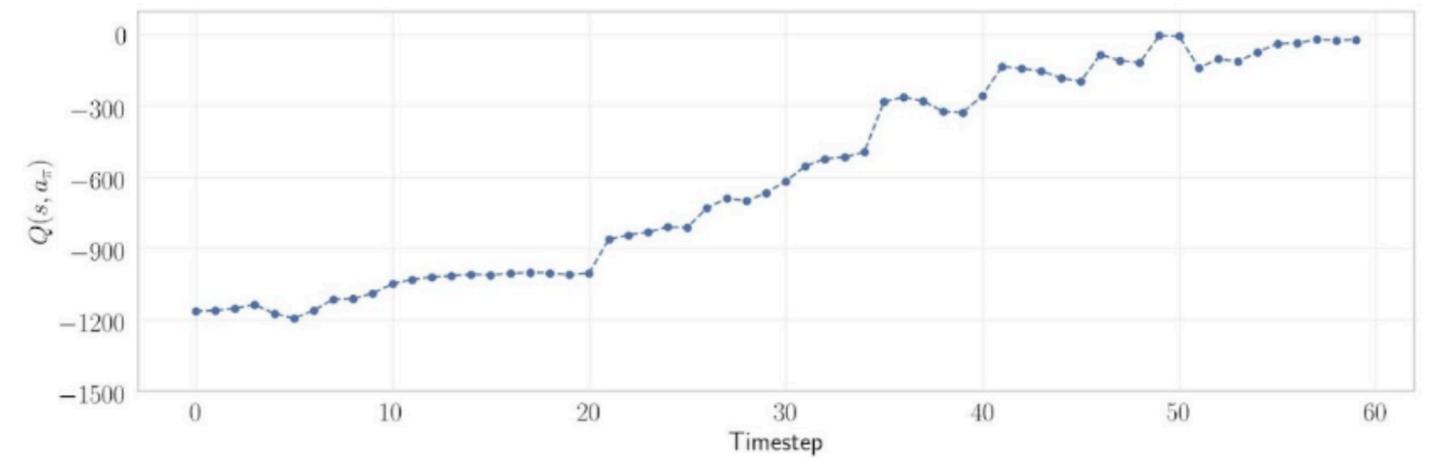
Moving towards parametric value functions

- A tabular value function intractable as state/action space grows!
- Learning a parametric value function can also allow for generalization
 - Related states have similar q-values (smoothness)
- Can use regression to learn value functions with a neural network



Discrete vs Continuous Q-values

- Different formulations for continuous and discrete action spaces:
 - In **continuous (a)**, typically condition on an action and output a scalar, where you sample actions from your policy for an expectation calculation
 - In **discrete (b)**, typically learn a value for each bin (output) and can enumerate over all actions for an expectation



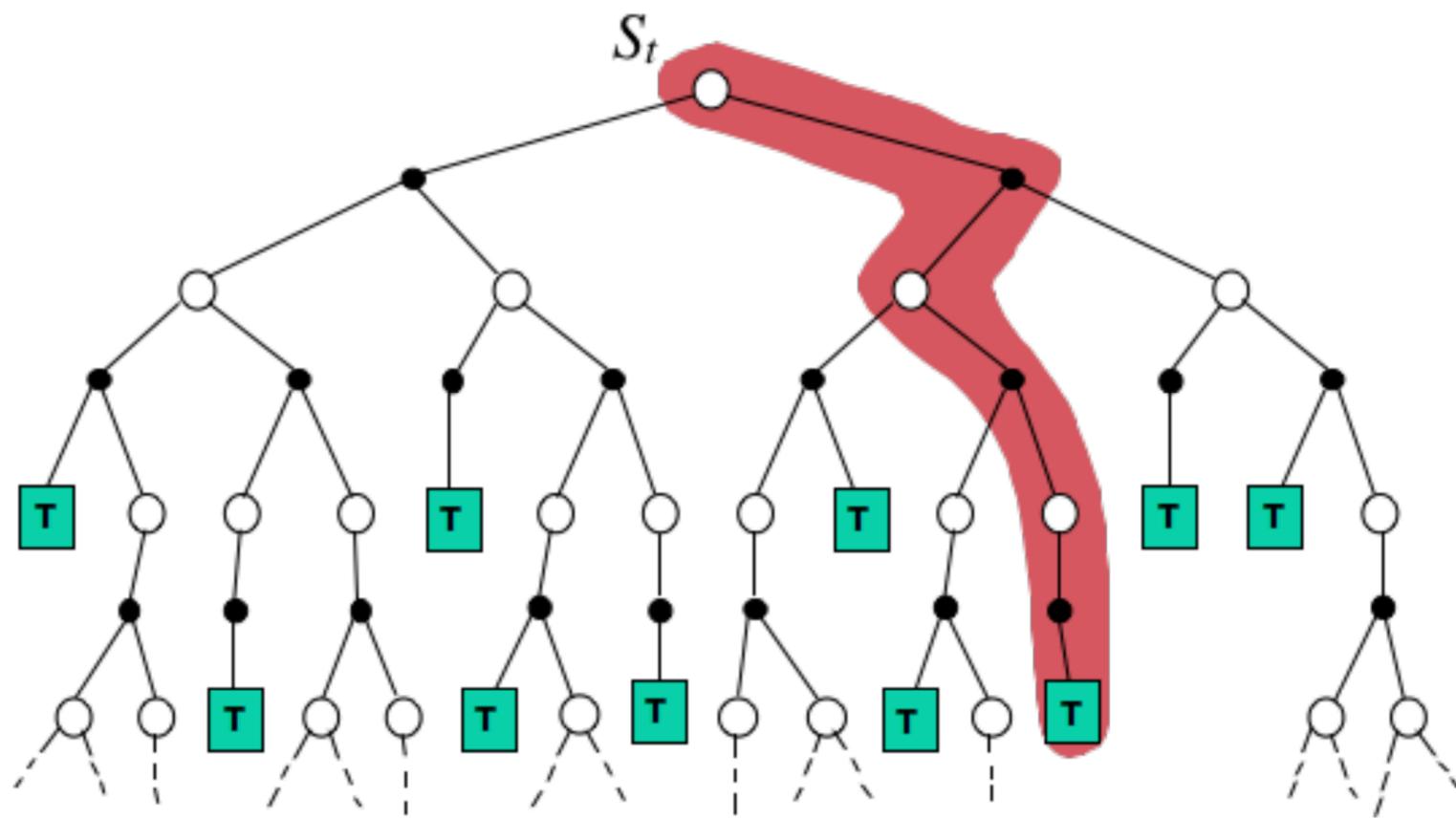
Q-Learning as a Regression Problem

Whiteboard

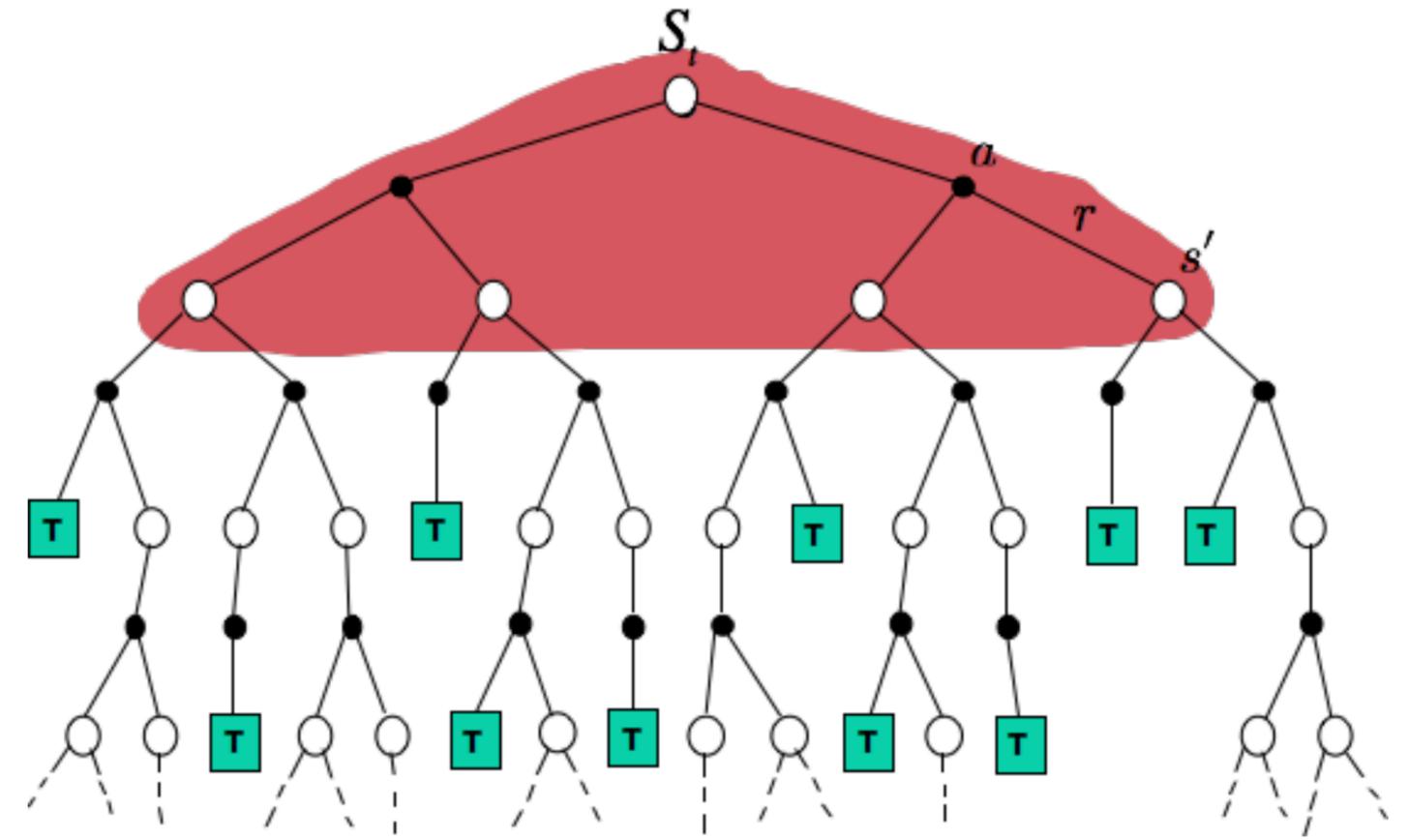
$$\begin{aligned} V^\pi(s) &= E_\pi [\mathcal{R}_t \mid S_t = s] && \text{(Definition)} \\ &= E_\pi \left[\sum_{k=0}^{\infty} \gamma^k r(s_{t+k}, a_{t+k}) \mid S_t = s \right] && \text{(Monte-Carlo Rollout, MC)} \\ &= E_\pi \left[r(s_t, a_t) + \gamma \sum_{k=0}^{\infty} \gamma^k r(s_{t+k+1}, a_{t+k+1}) \mid S_t = s \right] && \text{(Expansion)} \\ &= E_\pi [r(s_t, a_t) + \gamma V^\pi(s_{t+1}) \mid S_t = s] && \text{(Dynamic Programming, TD)} \\ &= E_\pi \left[r(s_t, a_t) + \gamma r(s_{t+1}, a_{t+1}) + \dots + \gamma^{k+1} V^\pi(s_{t+k+1}) \mid S_t = s \right] && \text{(N-Step)} \end{aligned}$$

Bias vs Variance Tradeoff!

MC vs TD Visualization



Monte-Carlo Rollouts
(Regression)



TD Backup
(Dynamic Programming)

Bias vs Variance Tradeoff

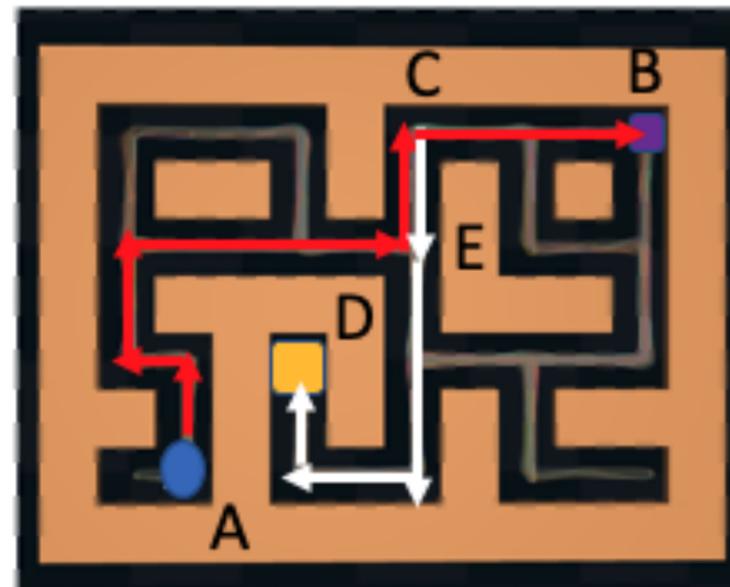
TD, MC, N-Step Return

Method	Bootstrap Depth	Bias	Variance
TD	1 Steps	High	Low
N-Step	N Steps	Medium	Medium
Monte Carlo	Full Episode	Low	High

Why Dynamic Programming?

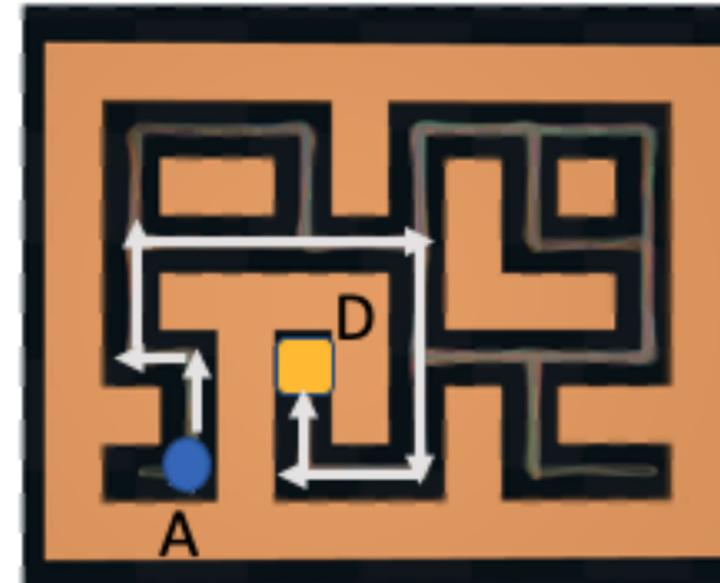
Aside on Stitching

Our maze navigation task



- Start (A) ■ Goal (D)
- trajectory from A → B
- trajectory from C → D

“Stitching” enables finding optimal behavior



Trajectory found by stitching segments of trajectories

Pause for Questions.

Practical Implementation Detail

Semi-Gradient and Target Network

Going back to our formulation:

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

Semi-Gradient

$$Q_{\theta}(s, a) \leftarrow r(s, a) + \gamma \max_{\hat{a}} \text{stopgrad} \left(Q_{\theta}(s', \hat{a}) \right)$$

Target Network

$$Q_{\theta}(s, a) \leftarrow r(s, a) + \gamma \max_{\hat{a}} \text{stopgrad} \left(Q_{\text{target}}(s', \hat{a}) \right)$$

Update Types

$$w' \leftarrow w \quad \text{when } \text{mod}(n, N) = 0$$

Hard Target Update

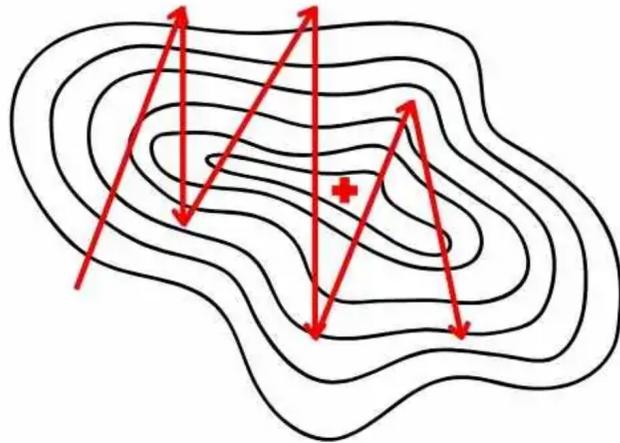
$$w' \leftarrow \tau \cdot w + (1 - \tau) \cdot w'$$

Soft Target Update (Polyak)

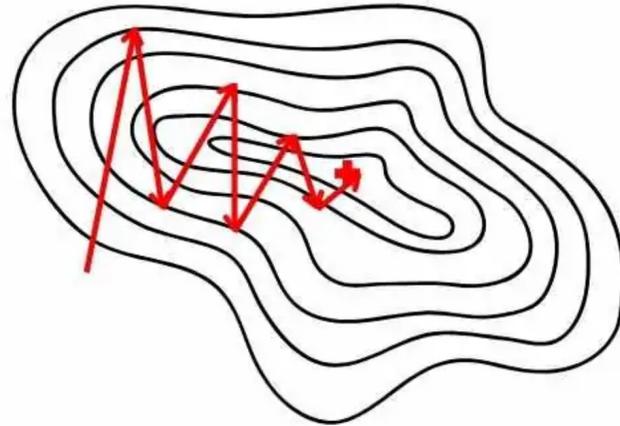
Practical Implementation Details

Dealing with TD Gradients

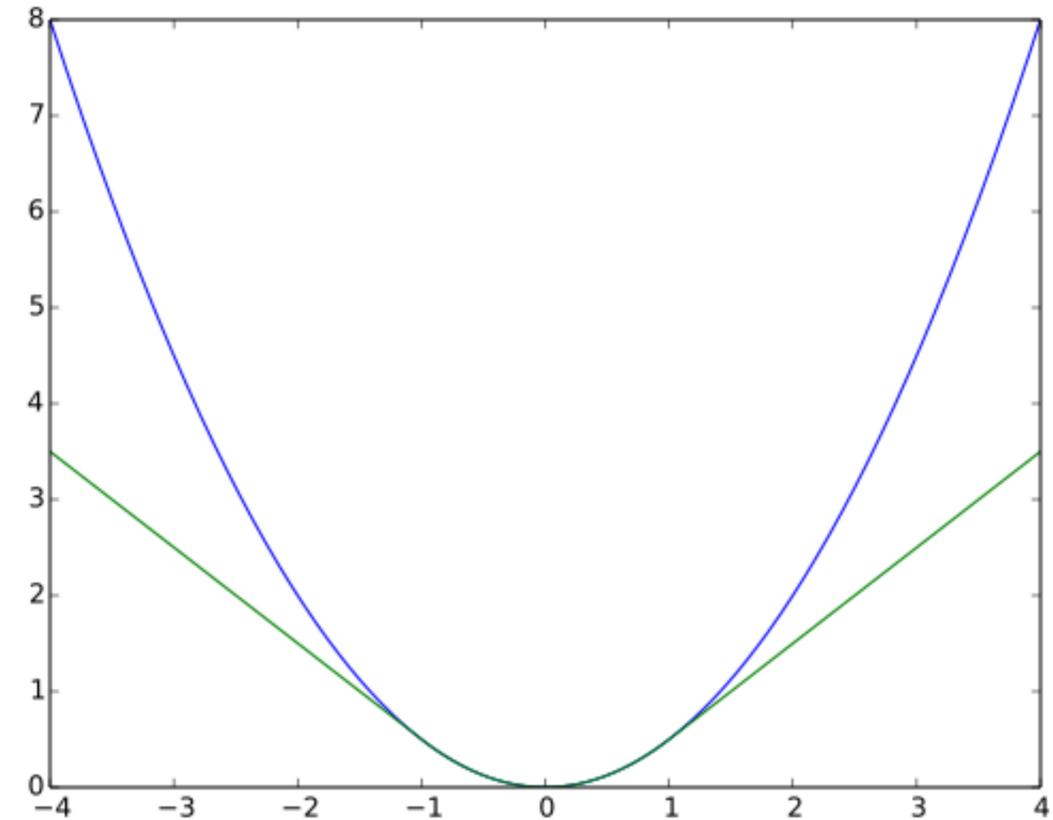
Without Gradient Clipping



With Gradient Clipping



Gradient Clipping

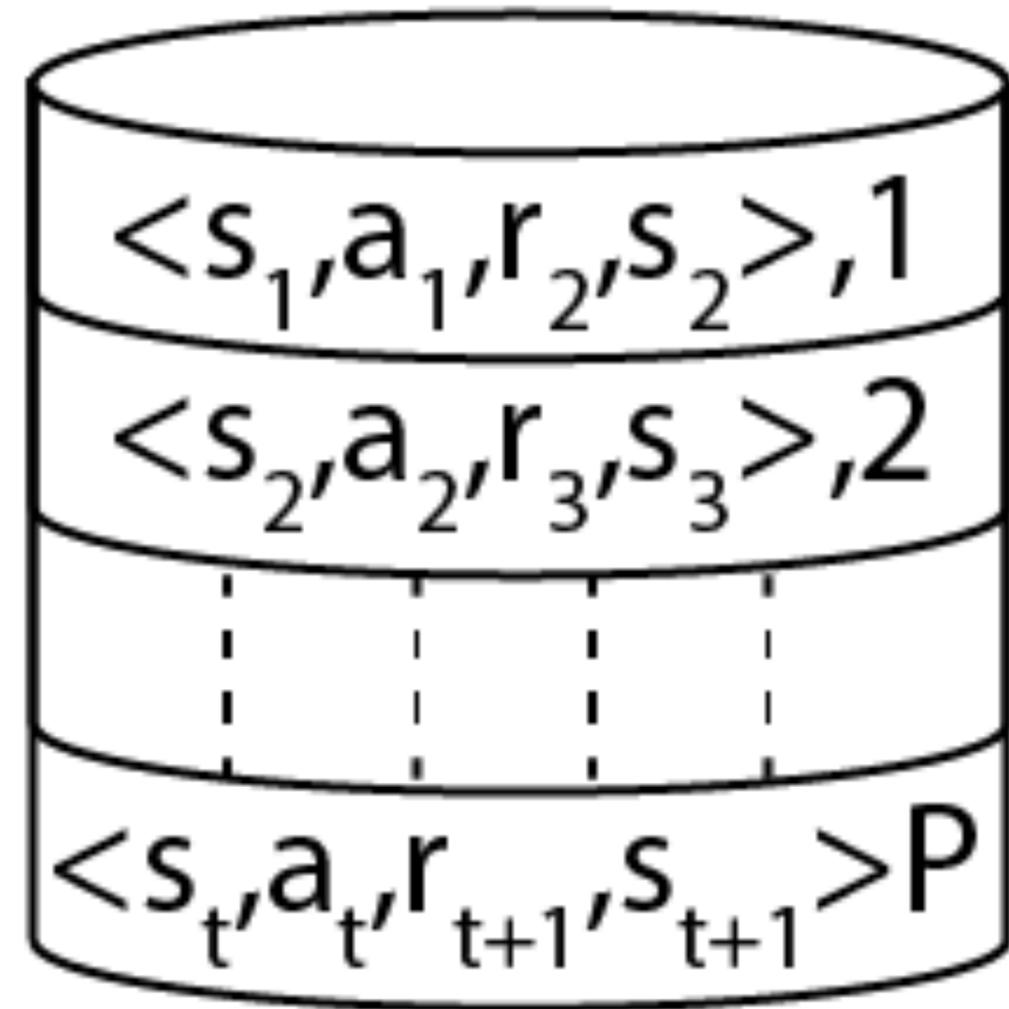


Huber Loss

Practical Implementation Details

Replay Buffers

- Replay Buffers are **memory structures** that store past experiences encountered by the agent, allowing the **experiences to be reused** instead of discarded after a single update
- They **break temporal correlations** between consecutive training samples and prevents recency bias
- **Improves sample efficiency** in Q-learning algorithms, allowing an agent to reuse its experience effectively



Q-Learning Overestimation

Q-Learning is typically formulated as follows:

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

When parameterized by a function approximator like a NN, there is some inherent noise in the q-value estimates due to modeling errors

$$Q^{\text{approx}}(s', \hat{a}) = Q^{\text{target}}(s', \hat{a}) + Y_{s'}^{\hat{a}}$$

Due to the noise on the RHS ($Y_{s'}^{\hat{a}}$), there results in error on the LHS, expressed as:

$$Z_s := \gamma \left(\max_{\hat{a}} Q^{\text{approx}}(s', \hat{a}) - \max_{\hat{a}} Q^{\text{target}}(s', \hat{a}) \right)$$

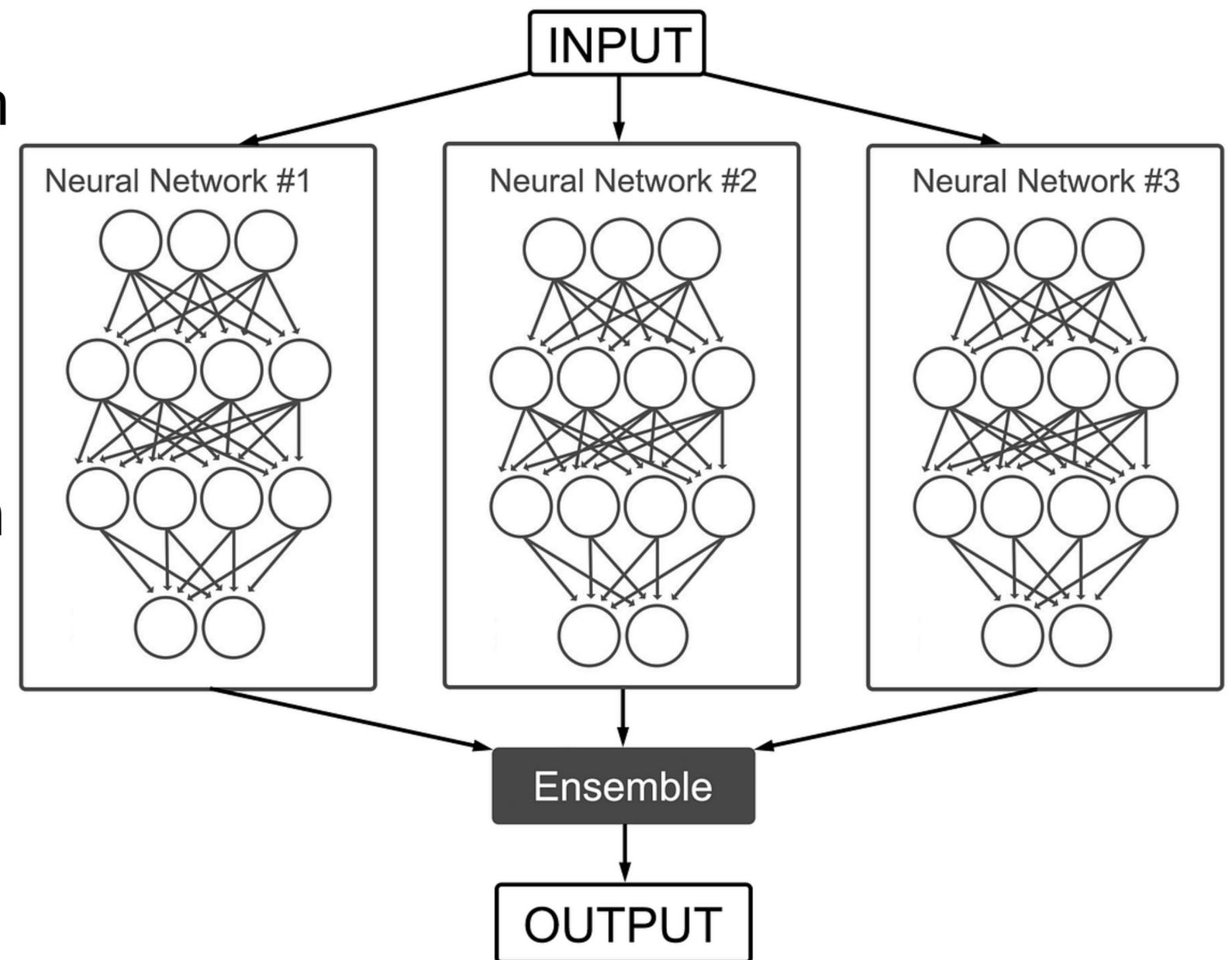
Even with zero mean ($Y_{s'}^{\hat{a}}$), Z_s may have positive mean:

$$\mathbb{E} [Y_{s'}^{\hat{a}}] = 0 \forall \hat{a} \implies \mathbb{E} [Z_s] > 0 \text{ (often)}$$

Practical Implementation Detail

Critic Ensembling

- One approach to tackle overestimation is to measure **epistemic uncertainty** with an ensemble of networks
- Since different model initializations may have **different modeling errors**, this can lead to ensembling with a **min reduction** to be effective
- Has been even shown to be effective in **offline RL** where distribution shift is a larger concern (Sac-N and EDAC)



Q-Learning vs Double Q-Learning

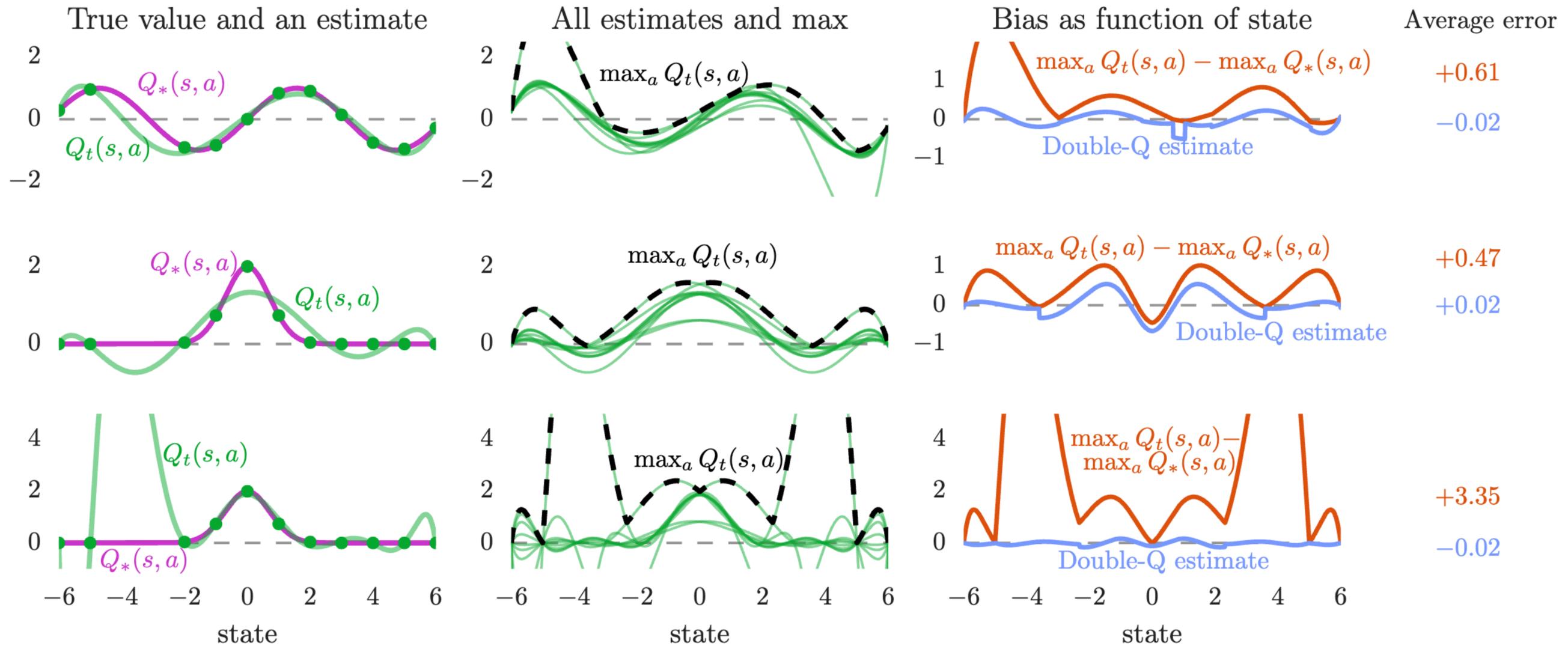


Figure 2: Illustration of overestimations during learning. In each state (x-axis), there are 10 actions. The **left column** shows the true values $V_*(s)$ (purple line). All true action values are defined by $Q_*(s, a) = V_*(s)$. The green line shows estimated values $Q(s, a)$ for one action as a function of state, fitted to the true value at several sampled states (green dots). The **middle column** plots show all the estimated values (green), and the maximum of these values (dashed black). The maximum is higher than the true value (purple, left plot) almost everywhere. The **right column** plots shows the difference in orange. The blue line in the right plots is the estimate used by Double Q-learning with a second set of samples for each state. The blue line is much closer to zero, indicating less bias. The three **rows** correspond to different true functions (left, purple) or capacities of the fitted function (left, green). (Details in the text)

Understanding HW Algorithms

Q-Learning : DQN

Whiteboard

Algorithm 1: DQN

Input: Replay buffer \mathcal{D} , Q-network Q_θ , target network $Q_{\theta'}$ with $\theta' \leftarrow \theta$, discount factor γ , batch size N , learning rate α , target update frequency C , exploration schedule ϵ

for *each interaction step* $t = 1, 2, \dots$ **do**

 Observe state s_t

 Choose action a_t using ϵ -greedy:

$$a_t = \begin{cases} \text{random action,} & \text{with probability } \epsilon, \\ \arg \max_a Q_\theta(s_t, a), & \text{otherwise} \end{cases}$$

 Execute a_t , observe (r_t, s_{t+1}) , and store (s_t, a_t, r_t, s_{t+1}) in \mathcal{D}

 // Sample a mini-batch

 Sample $\{(s_i, a_i, r_i, s_{i+1})\}_{i=1}^N \sim \mathcal{D}$

 // Compute targets

for $i = 1, \dots, N$ **do**

$y_i \leftarrow r_i + \gamma \max_{a'} Q_{\theta'}(s_{i+1}, a')$

 // Q-network update

$$\theta \leftarrow \theta - \alpha \nabla_\theta \frac{1}{N} \sum_{i=1}^N [Q_\theta(s_i, a_i) - y_i]^2$$

 // Periodic target update

if $t \bmod C = 0$ **then**

$\theta' \leftarrow \theta$

Actor Critic: SAC

Whiteboard

Algorithm 1: Soft Actor-Critic (no entropy term)

Input: Replay buffer \mathcal{D} , critic nets $Q_{\theta_1}, Q_{\theta_2}$, actor net π_ϕ , target critics $\theta'_i \leftarrow \theta_i$, discount γ , smoothing τ , batch size N

for each interaction step $t = 1, 2, \dots$ **do**

 Observe state s_t

 Sample action $a_t \sim \pi_\phi(\cdot | s_t)$

 Execute a_t , observe (r_t, s_{t+1}) and store in \mathcal{D}

 // Sample a mini-batch

 Sample $\{(s_i, a_i, r_i, s_{i+1})\}_{i=1}^N \sim \mathcal{D}$

 // Compute targets (no entropy term)

for $i = 1, \dots, N$ **do**

$a'_{i+1} \leftarrow \pi_\phi(s_{i+1});$
 $y_i \leftarrow r_i + \gamma \min_{j=1,2} Q_{\theta'_j}(s_{i+1}, a'_{i+1})$

 // Critic update

for $j = 1, 2$ **do**

$\theta_j \leftarrow \theta_j - \lambda_Q \nabla_{\theta_j} \frac{1}{N} \sum_i [Q_{\theta_j}(s_i, a_i) - y_i]^2$

 // Actor update (maximize Q only)

$\phi \leftarrow \phi + \lambda_\pi \nabla_\phi \frac{1}{N} \sum_i Q_{\theta_1}(s_i, \pi_\phi(s_i))$

 // Target networks soft-update

for $j = 1, 2$ **do**

$\theta'_j \leftarrow \tau \theta_j + (1 - \tau) \theta'_j$

Thanks!