

DEVELOPING A SIMPLE REINFORCEMENT LEARNING PROGRAM USING RL-GLUE

Scott O'Hara

Metrowest Developers Machine Learning Group

09/02/2020

REFERENCES

Sample-based Learning Methods (M. White and A. White), University of Alberta, Alberta Machine Intelligence Institute, Coursera.

► <https://www.coursera.org/learn/sample-based-learning-methods/>

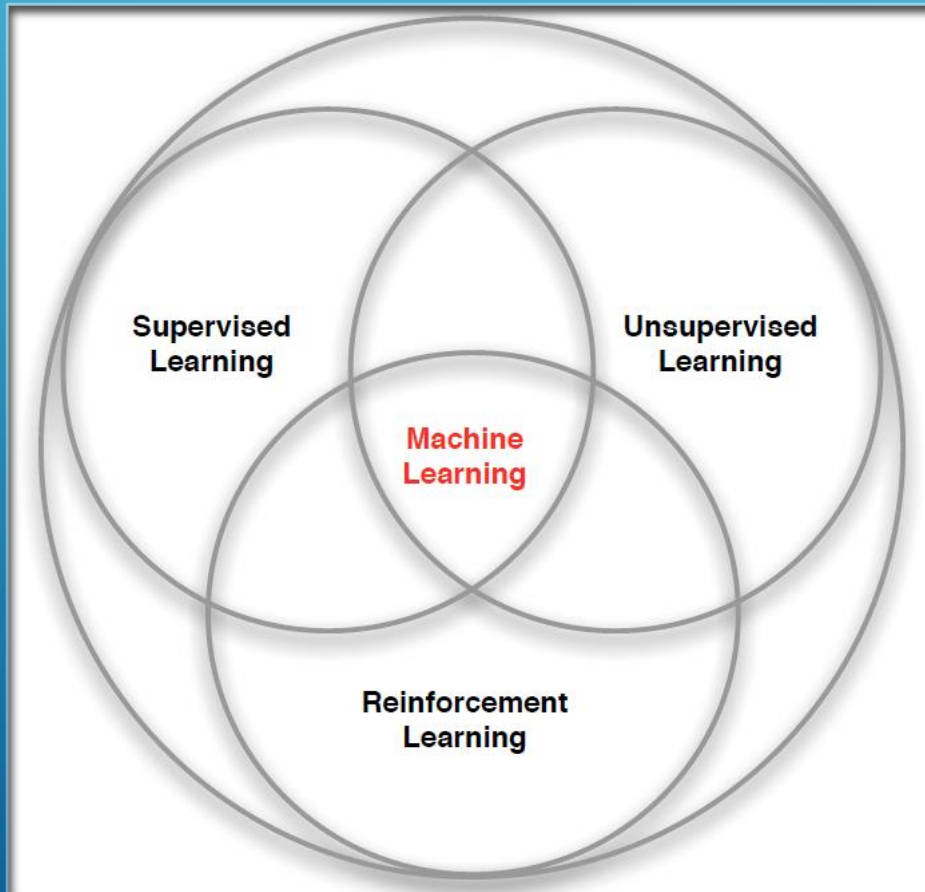
Reinforcement learning: An Introduction R. S. Sutton and A. G. Barto, Second edition. Cambridge, Massachusetts: The MIT Press, 2018.

Reinforcement Learning David Silver - University College London / Google DeepMind, 2015.

► <https://www.davidsilver.uk/teaching/>

WHAT IS REINFORCEMENT LEARNING (RL)?

“Reinforcement learning is a kind **of *unsupervised supervised learning***”
– Rich Sutton




Supervised Learning – Learn a function from labeled data that maps input attributes to an output.


Unsupervised Learning – Find classes, patterns or generalizations in unlabeled data.

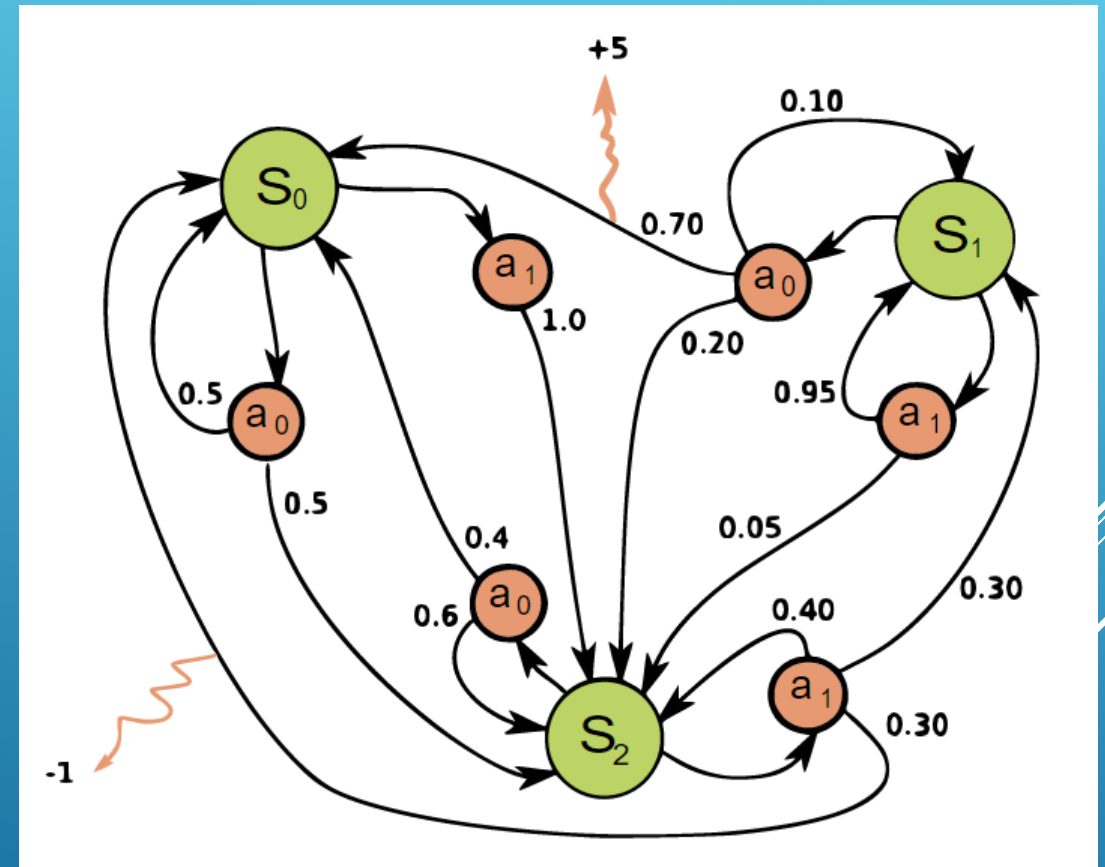
Reinforcement Learning – An agent learns to maximize rewards while acting in an uncertain environment.

RL PROBLEMS ARE OFTEN MODELED WITH A MARKOV DECISION PROCESS (MDP)

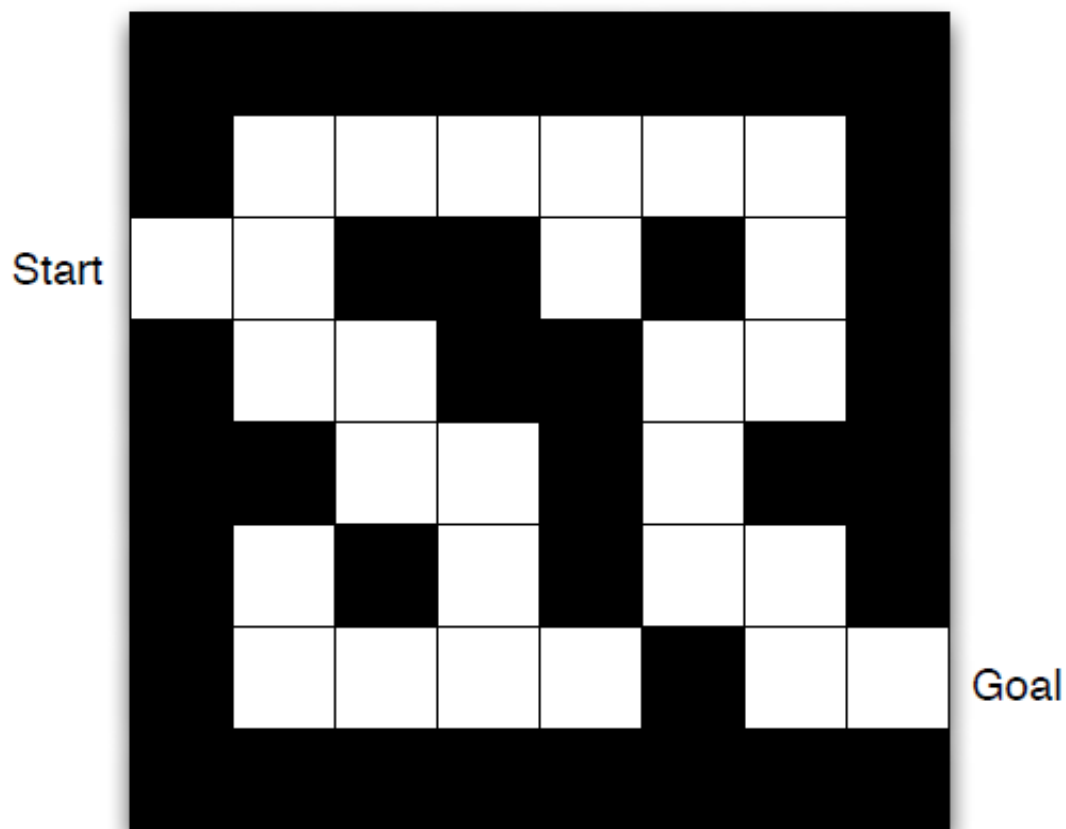
- **States:** s_1, \dots, s_n
- **Actions:** a_1, \dots, a_m
- **Reward model:**  +5

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

- **Transition model:**  0.5
 $T(s, a, s') = P(s'|s, a)$

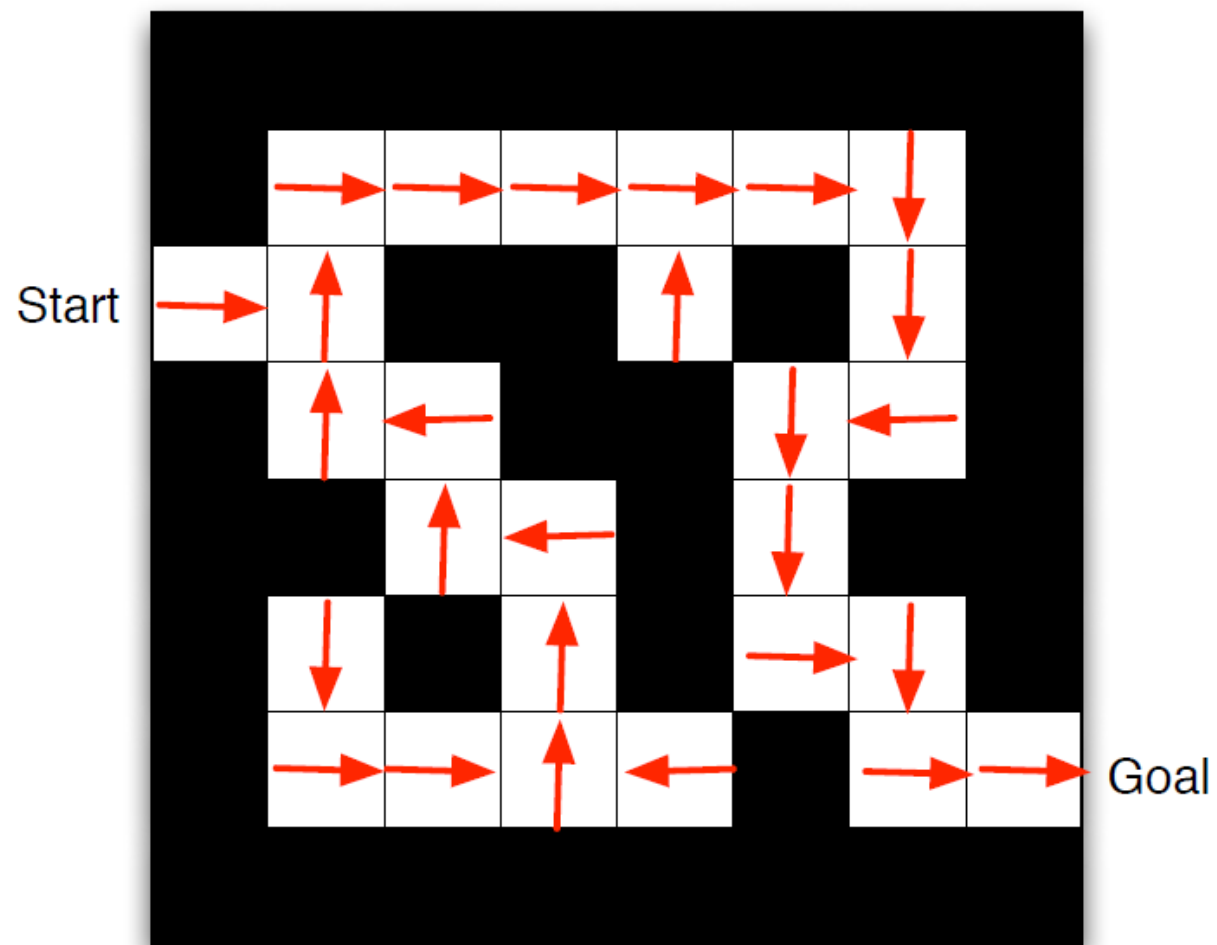


Maze Example



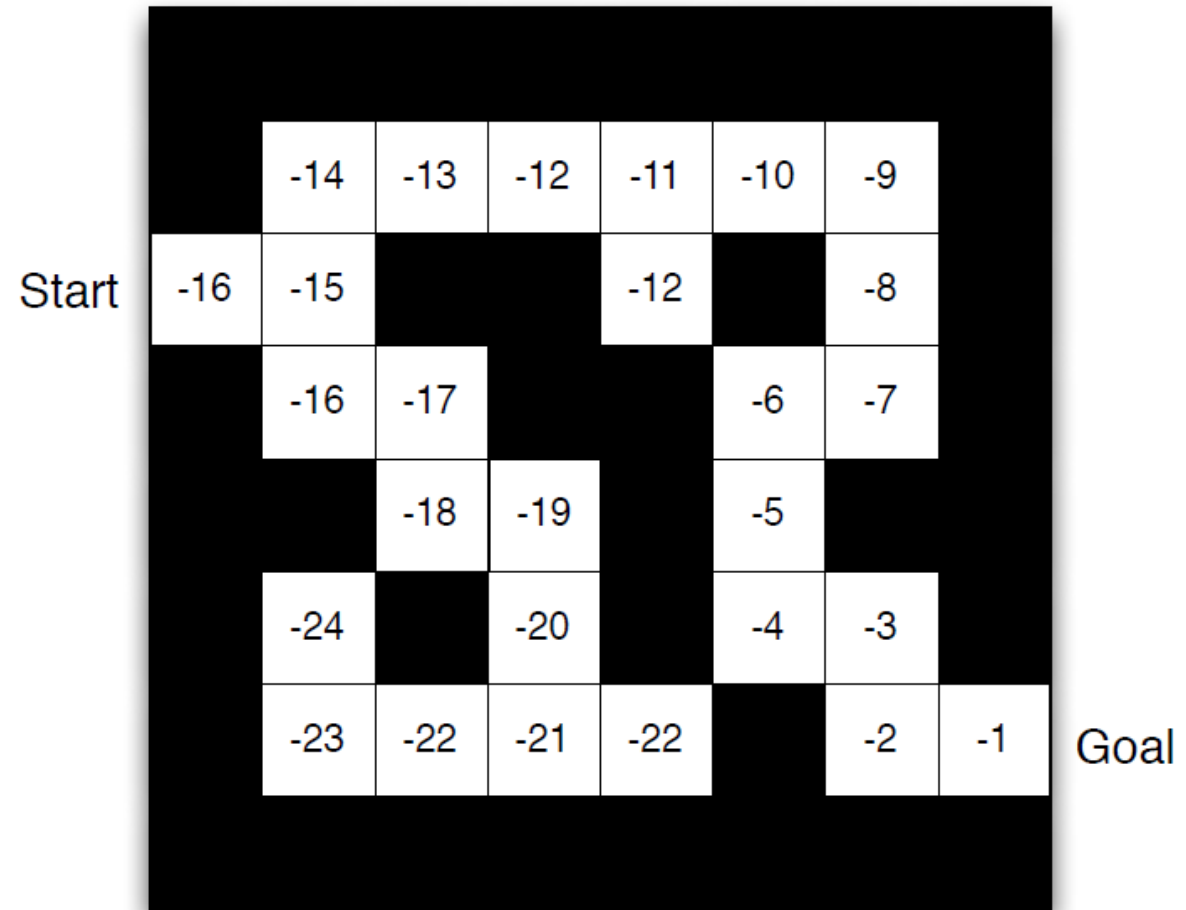
- Rewards: -1 per time-step
- Actions: N, E, S, W
- States: Agent's location

Maze Example: Policy



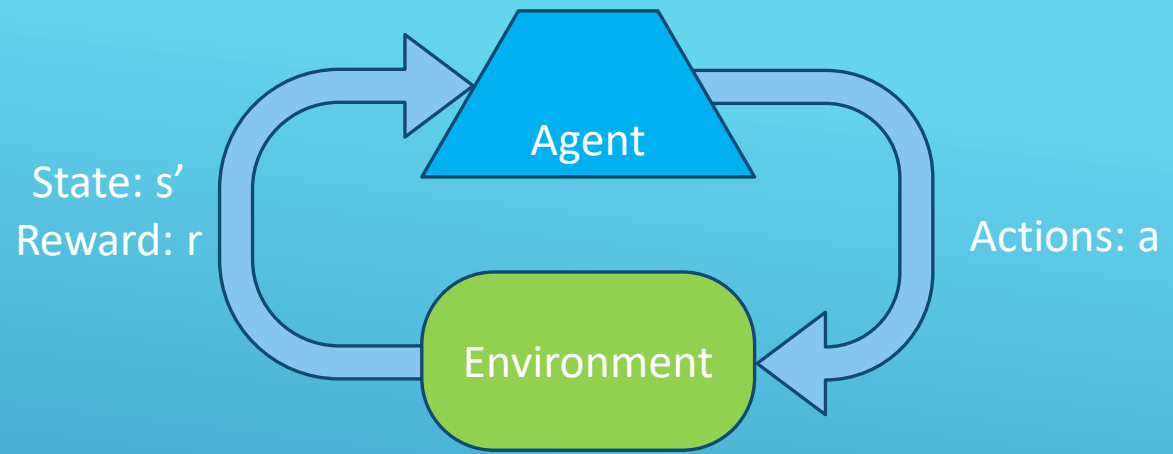
■ Arrows represent policy $\pi(s)$ for each state s

Maze Example: Value Function



- Numbers represent value $v_{\pi}(s)$ of each state s

THE REINFORCEMENT LEARNING PROBLEM



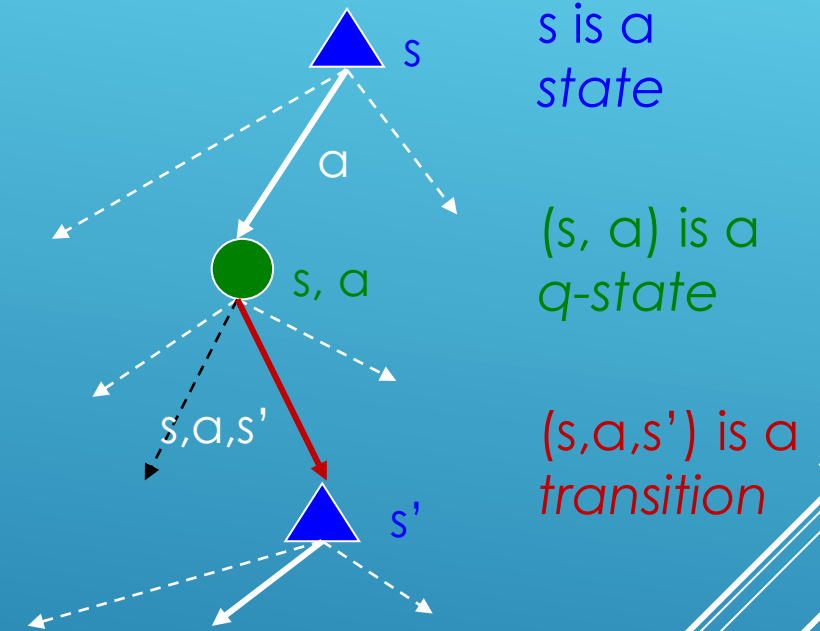
- Agent must learn to act to maximize expected rewards in the environment.
- Agent knows the current state \mathbf{s} , takes an action \mathbf{a} , receives a reward \mathbf{r} and observes the next state \mathbf{s}' .

$\mathbf{S}_0, \mathbf{A}_0, \mathbf{R}_0, \mathbf{S}_1, \mathbf{A}_1, \mathbf{R}_1, \mathbf{S}_2, \mathbf{A}_2, \mathbf{R}_2, \dots, \mathbf{S}_n, \mathbf{A}_n, \mathbf{R}_n, \mathbf{S}_T$

REINFORCEMENT LEARNING ALGORITHMS

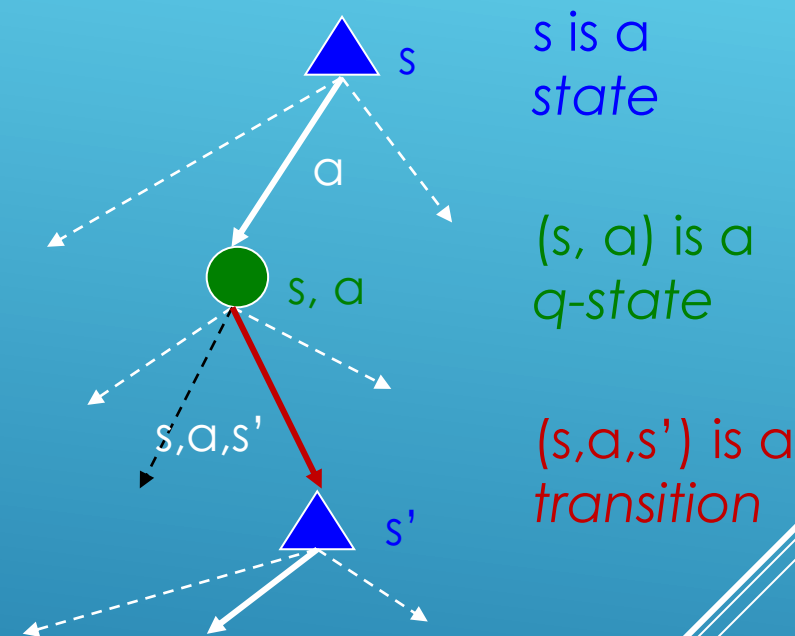
OPTIMIZE VALUE FUNCTIONS V AND Q

- The State-Value Function:
 $V(s)$ = expected utility of starting in state s and acting optimally thereafter.
- The Action-Value Function:
 $Q(s,a)$ = expected utility of starting in state s , taking action a , and acting optimally thereafter.



MODEL-FREE RL ALGORITHMS

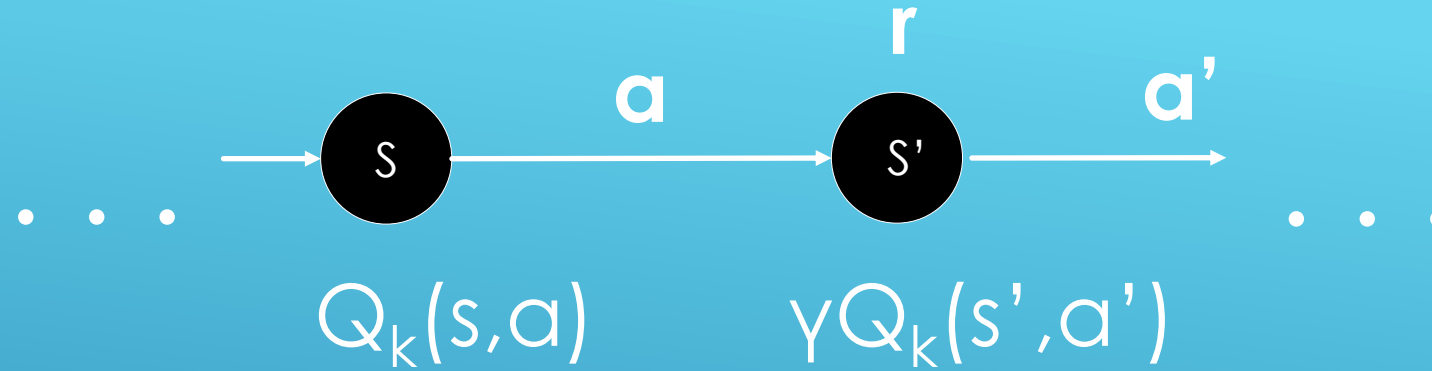
- The state-value function $V(s)$ can be used to make optimal decisions only when used with the transition function $T(s, a, s')$ and the reward function $R(s, a, s')$.
- Fortunately, the action-value function $Q(s, a)$ can be computed directly without the transition and reward functions.



MODEL-FREE ALGORITHMS

- RL algorithms that compute the Q function directly are called ***model-free algorithms***.
- Sarsa and Q-Learning are two examples of model-free algorithms that use **temporal difference (TD) learning** to compute the Q function directly.
- With **TD-Learning**:
 - You make a prediction about what will happen next.
 - You wait to see what happens.
 - You learn by comparing what happens to what you predicted and adjusting your decision policy accordingly.

SARSA Update Rule



$$Q_{k+1}(s, a) \leftarrow Q_k(s, a) + \alpha [r + \gamma Q_k(s', a') - Q_k(s, a)]$$

γ – discount rate.
 α – learning rate.

The Future
(The Target)

The Present
(The Prediction)

SARSA Algorithm

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

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 $Q(\text{terminal}, \cdot) = 0$

Loop for each episode:

 Initialize S

 Choose A from S using policy derived from Q (e.g., ε -greedy)

 Loop for each step of episode:

 Take action A , observe R, S'

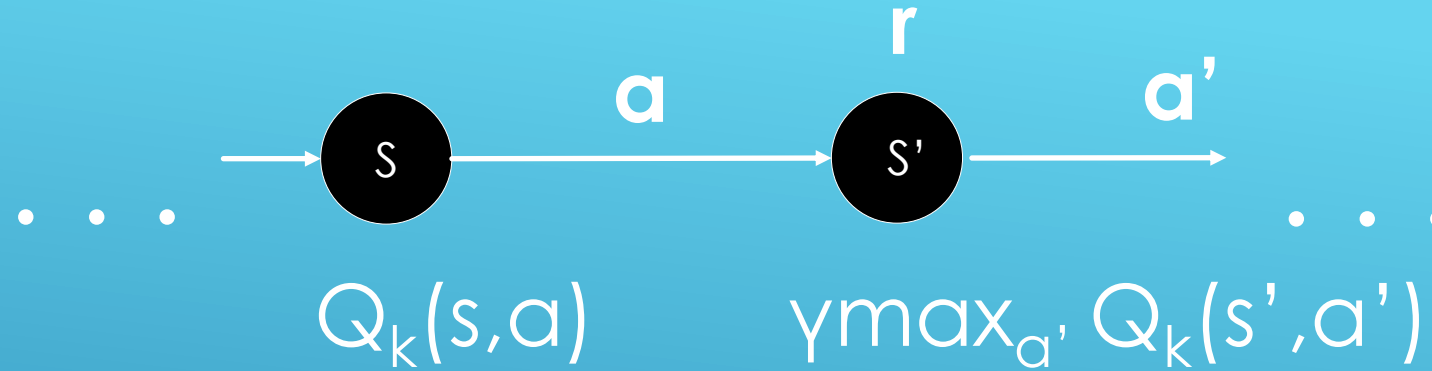
 Choose A' from S' using policy derived from Q (e.g., ε -greedy)

$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma Q(S', A') - Q(S, A)]$

$S \leftarrow S'; A \leftarrow A';$

 until S is terminal

Q-Learning Update Rule



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

γ – discount rate.
 α – learning rate.

The Future
(The Target)

The Present
(The Prediction)

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 $Q(\text{terminal}, \cdot) = 0$

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)

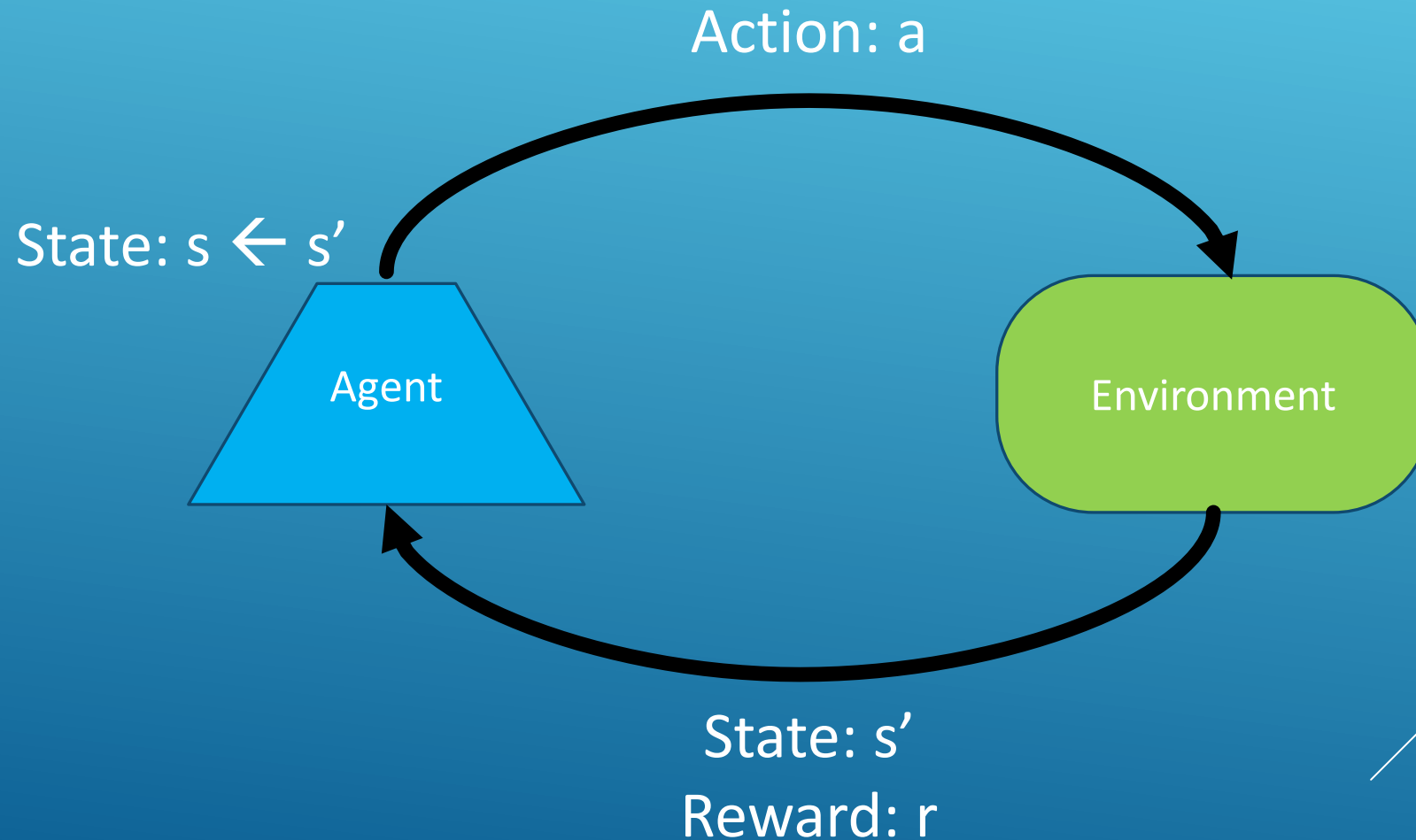
 Take action A , observe R, S'

$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$

$S \leftarrow S'$

 until S is terminal

How do we Implement the Reinforcement Learning Loop?



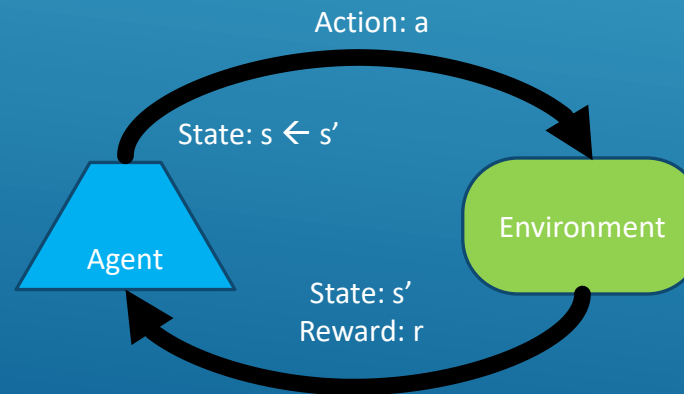
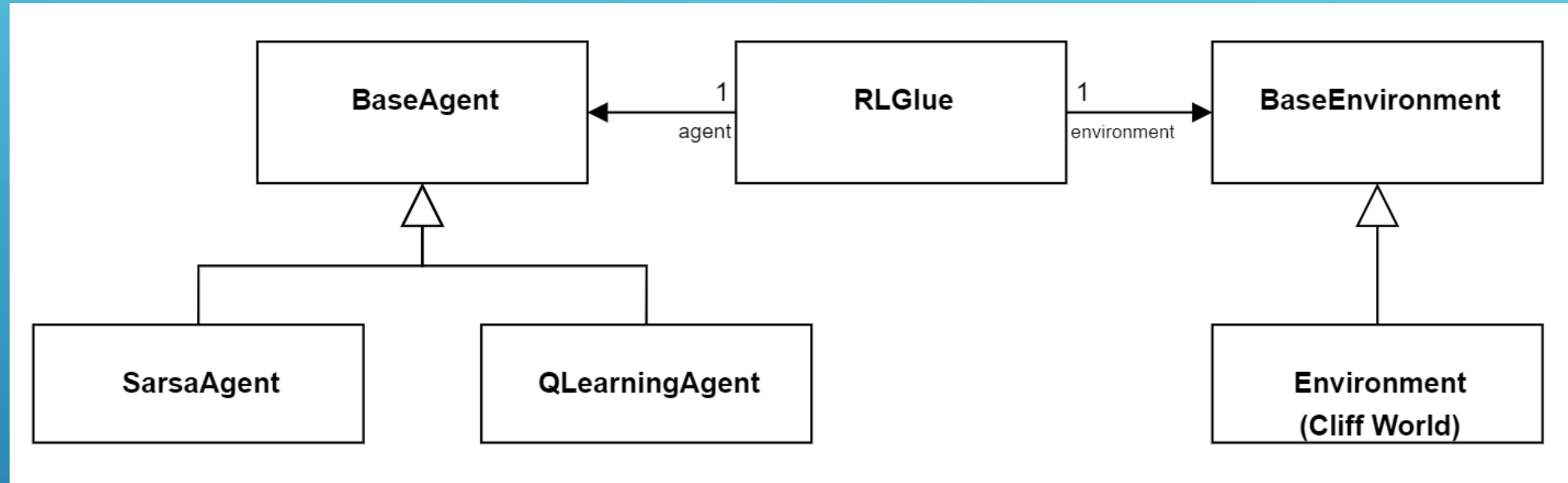
A SIMPLE APPROACH: RL-GLUE

- 3 Python Classes:
 - RLGlue
 - BaseEnvironment
 - BaseAgent
- In 3 Files:
 - rl_glue.py
 - environment.py
 - agent.py
- Obtained from the ***Reinforcement Learning Specialization*** on Coursera.

RL-GLUE (HISTORIC)

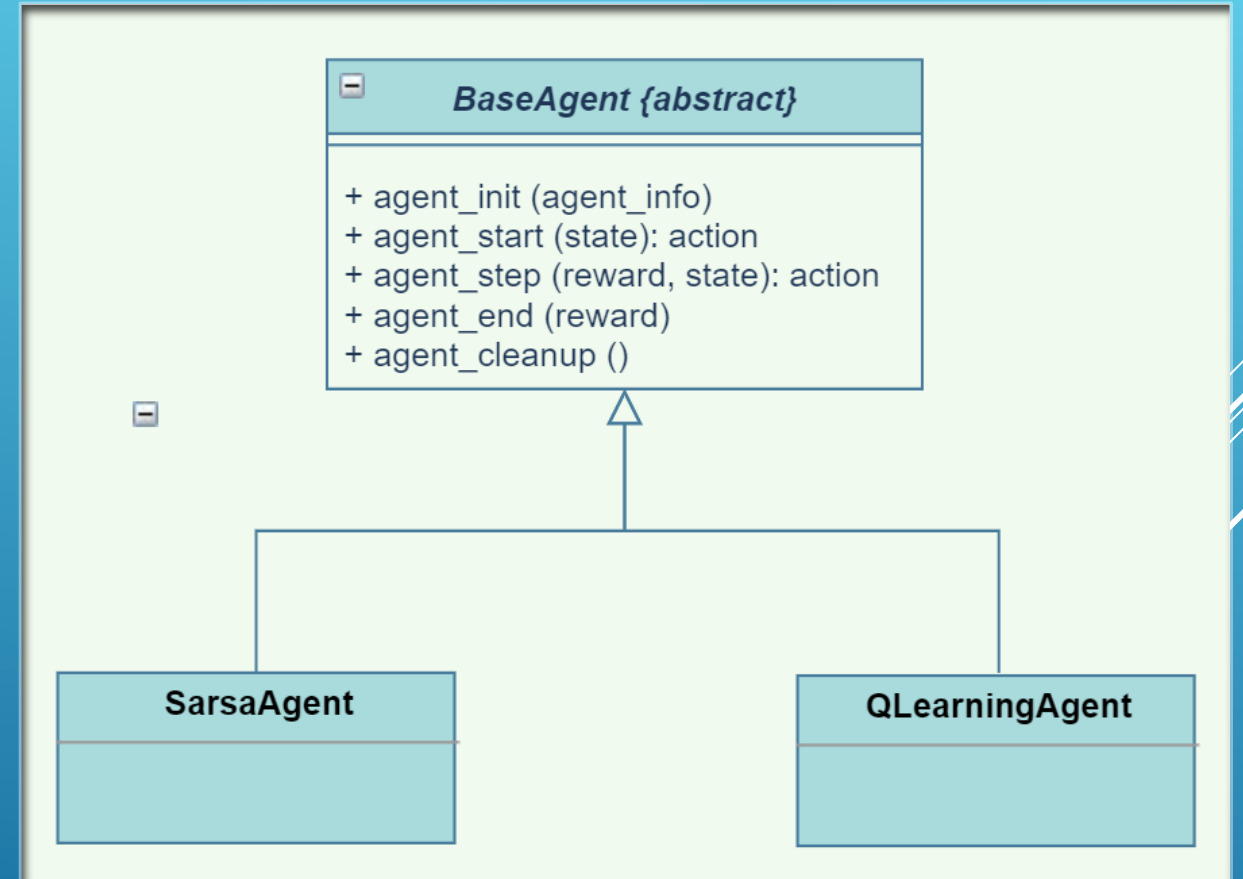
- Originally developed by Brian Tanner and Adam White, colleagues of Rich Sutton, author of Reinforcement Learning: An Introduction.
- Some old links, most obsolete, some broken.
 - <http://incompleteideas.net/rlai.cs.ualberta.ca/RLAI/rlai.html>
 - <https://sites.google.com/a/rl-community.org/rl-glue>
 - <https://code.google.com/archive/p/rl-glue/>
- Another version of RL-Glue in Python
 - NOT the Coursera version.
 - <https://github.com/amarack/python-rl>

Implementing the RL Loop with RLGlue



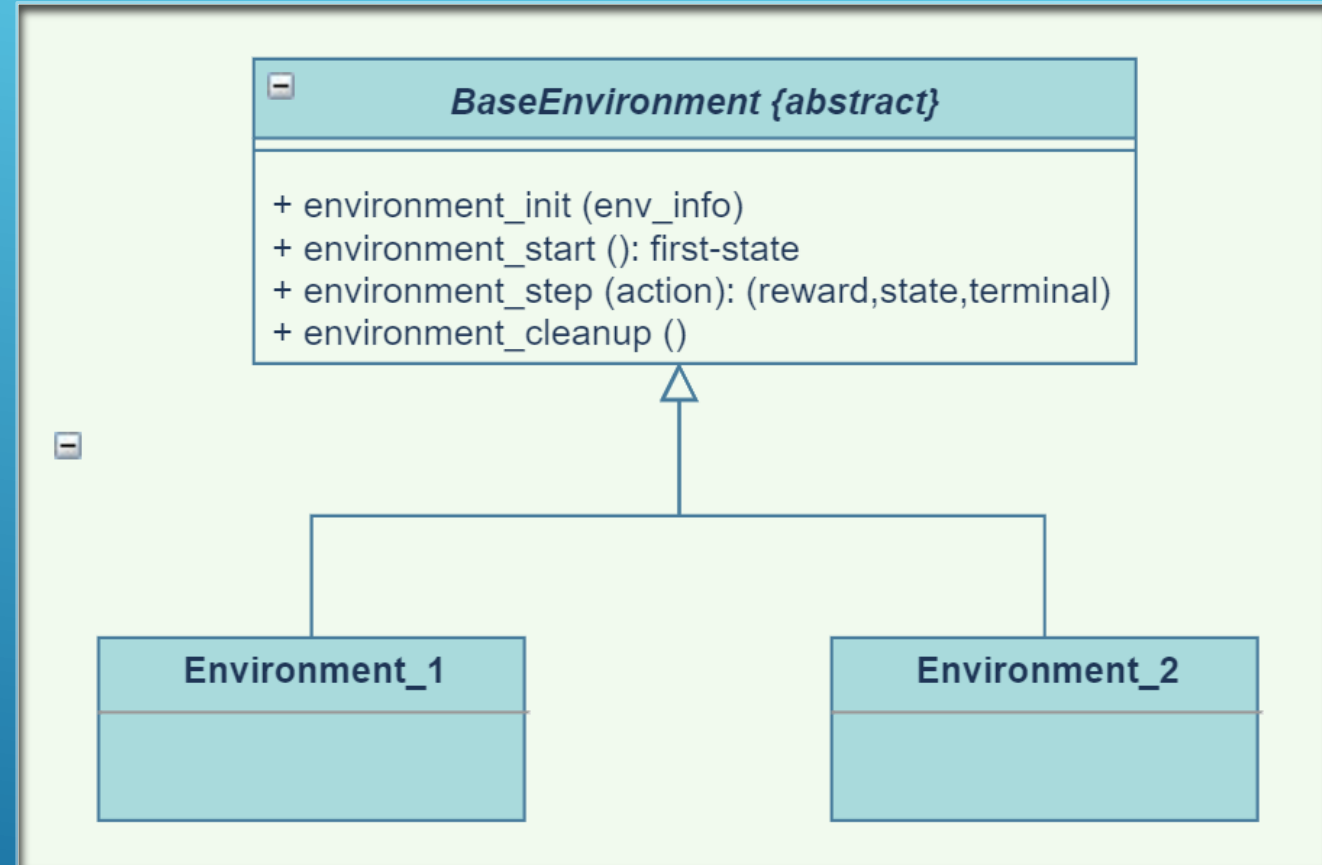
THE BASE AGENT CLASS

- **agent_init (info)** – setup agent when experiment first starts.
- **agent_start(state): action** – process first state from environment and return an action.
- **agent_step (reward, state): action** – process reward and state from environment and return an action.
- **agent_end (reward)** – process final reward after terminal state is reached.
- **agent_cleanup ()** – cleanup after agent is finished.



THE BASE ENVIRONMENT CLASS

- **environment_init (info)** – setup environment when experiment first starts.
- **environment_start(): first-state** – return initial state.
- **environment_step (action): (reward,state,terminal)** – given an action, return a reward, the next state, and a Boolean indicating whether the state is terminal.
- **environment_cleanup ()** – cleanup after agent is finished.



THE RLGLUE CLASS

RLGlue
<ul style="list-style-type: none">+ <code>__init__(Agent, Environment)</code>+ <code>rl_init (env_info)</code>+ <code>rl_start ()</code>: first-state+ <code>rl_step ()</code>: (reward,state,action,terminal)+ <code>rl_episode()</code>+ <code>rl_cleanup ()</code>

- **`__init__(Agent, Environment)`** – RLGlue creates an agent object and an environment object.
- **`rl_init(agent_init_info, env_init_info)`** – setup environment and agent.
- **`rl_start(): first-state`** – environment sets the initial state; agent takes its first action; return initial state.
- **`rl_step(): (reward,state,action,term)`** – environment changes in response to the agent's action; agent acts in response.
- **`rl_episode()`** - run an episode.
- **`rl_cleanup()`** – cleanup after agent is finished.

RL-GLUE SEQUENCE DIAGRAM

__init__()

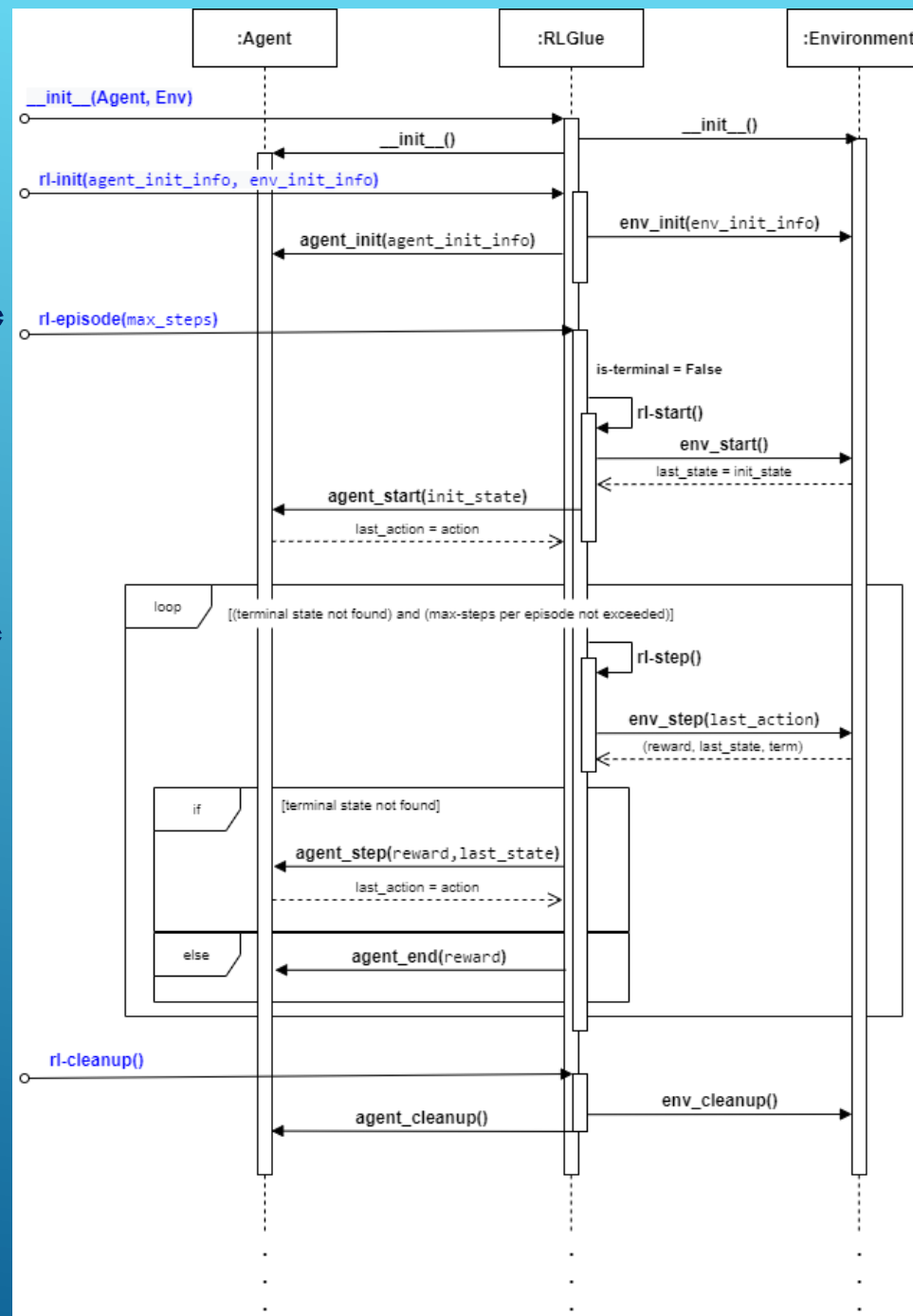
rl-init()

rl-episode()*

rl-start()

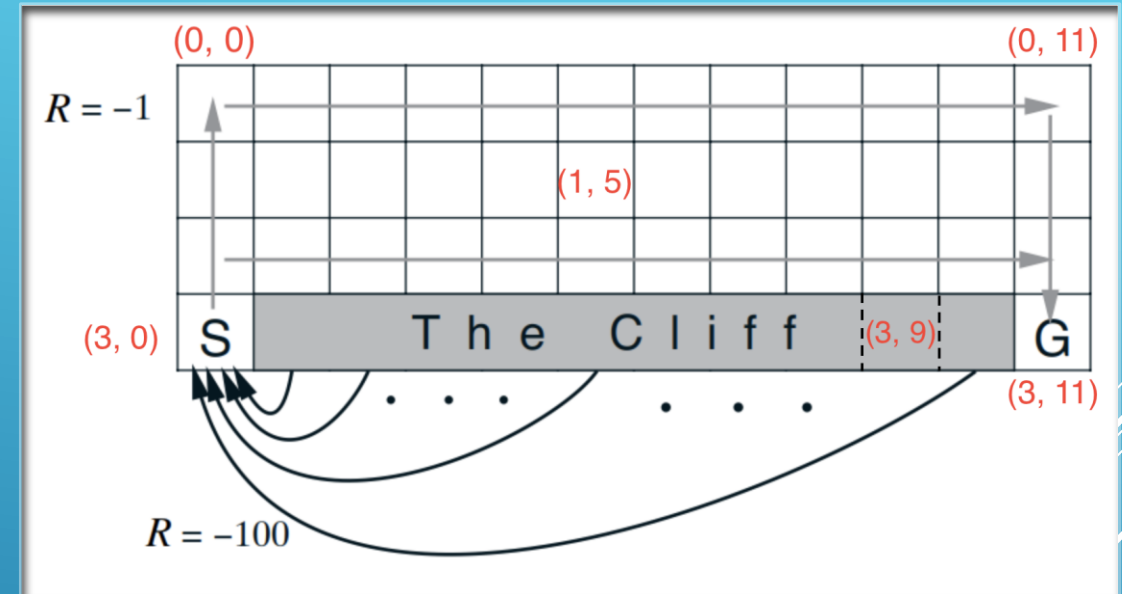
rl-step()*

rl-cleanup()



RL ENVIRONMENT: CLIFFWORLD

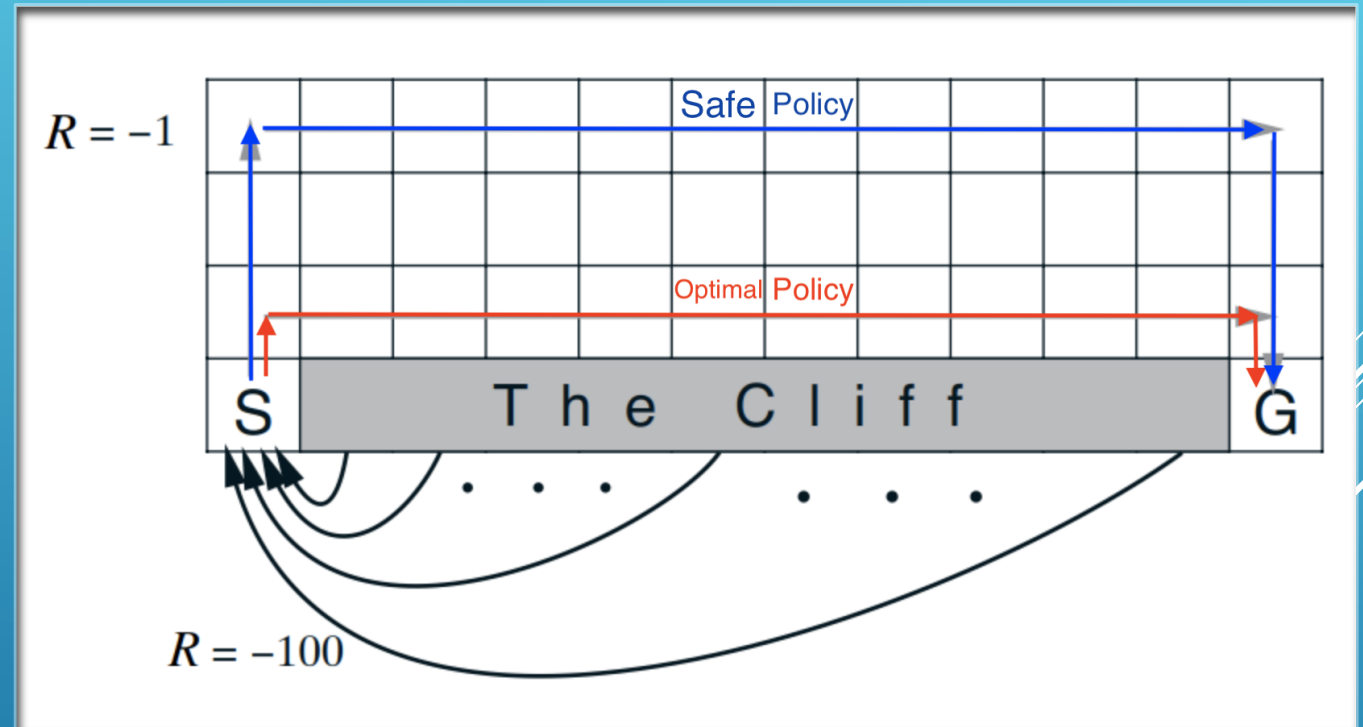
- **States:** $\langle x, y \rangle$ locations
- **Actions:** move **north**, **south**, **east** or **west**.
- **Reward model:**
 - 0 if robot moves to the goal state G where episode finishes.
 - -100 if robot moves to the cliff.
 - -1 for every other move.
- **Transition model:**
 - Robot moves deterministically in the chosen direction: **north**, **south**, **east** or **west**.
 - Robot stays put if it moves into a wall.
 - Robot transitions to the start state if it moves onto the cliff. (NOTE: episode does not finish.)



- **Discount factor:** $\gamma = 1.0$

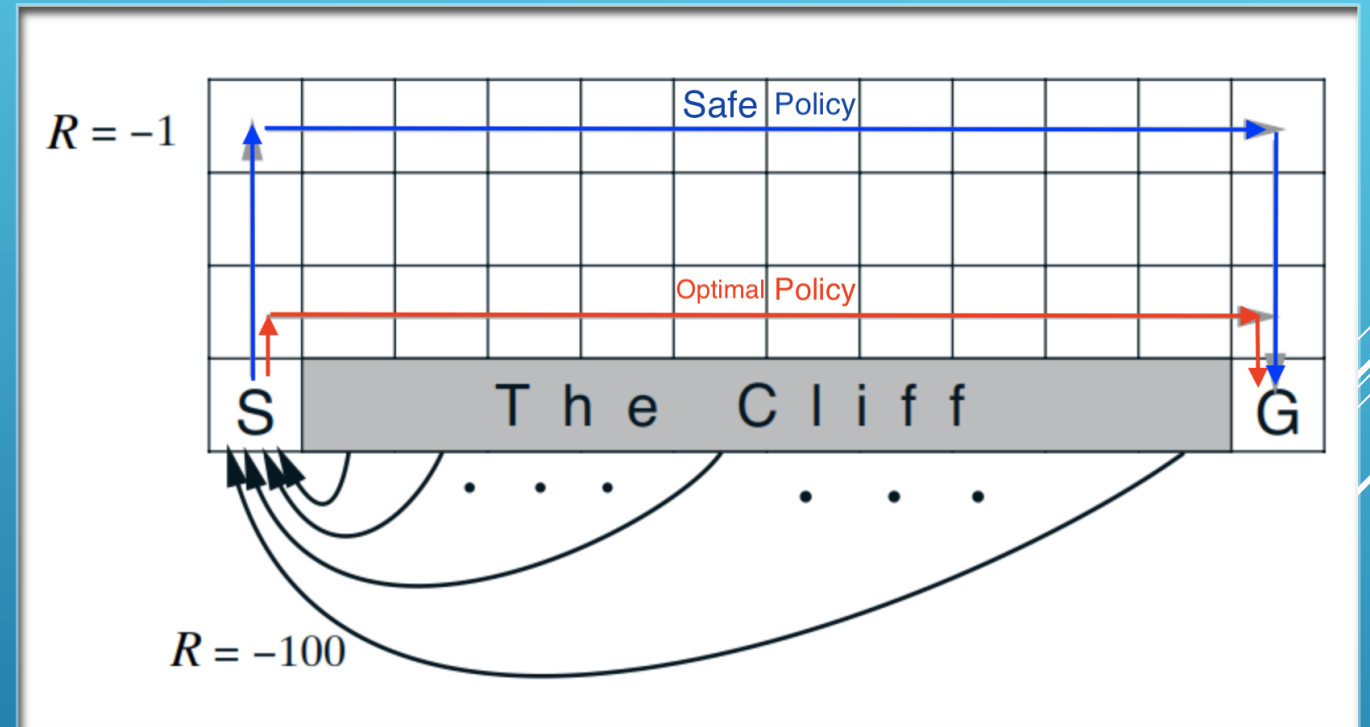
OPTIMAL VS. “SAFE” POLICIES: Q-LEARNING

- Q-learning learns the optimal policy.
- However, q-learning must stop exploring and change to complete exploitation mode to take advantage of this.
- If q-learning continues to explore (off-policy), it will often get bad results since exploration will lead it to step over the cliff.



OPTIMAL VS. “SAFE” POLICIES: SARSA

- SARSA learns the safe policy since it learns the policy it actually does.
- However, SARSA learns slowly since it doesn't take full advantage of the knowledge it has of state-action values.



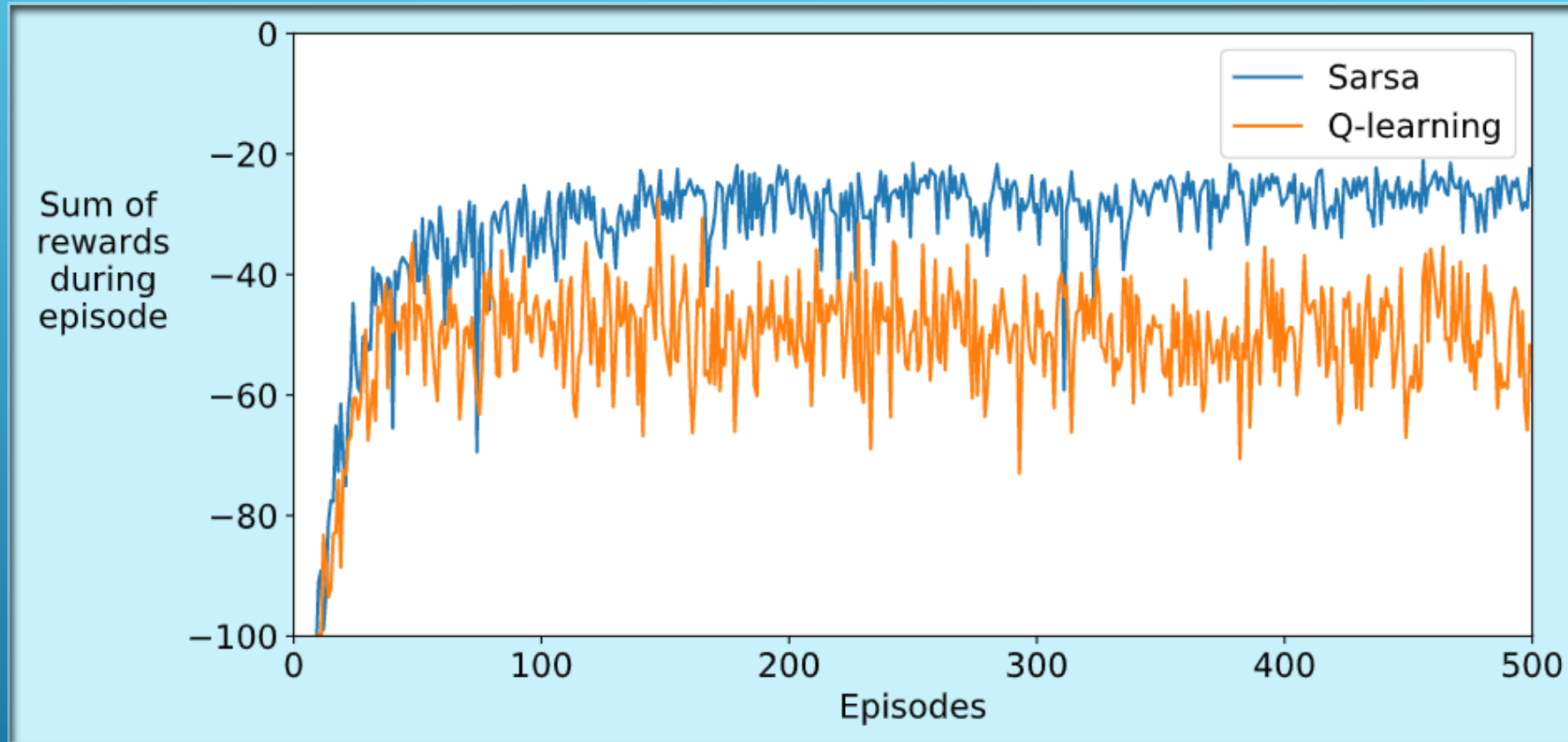
CHOOSING AN ACTION: EXPLORATION VS EXPLOITATION

- 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.

LOOK AT THE FOLLOWING FILES:

- agent.py
 - environment.py
 - rl_glue.py
 - cliffworld_env.py
 - cliffwalk_annotated.png
 - Simple-RL-Glue-Demo.ipynb
- 
- A series of white diagonal lines of varying lengths and thicknesses, located in the bottom right corner of the slide.

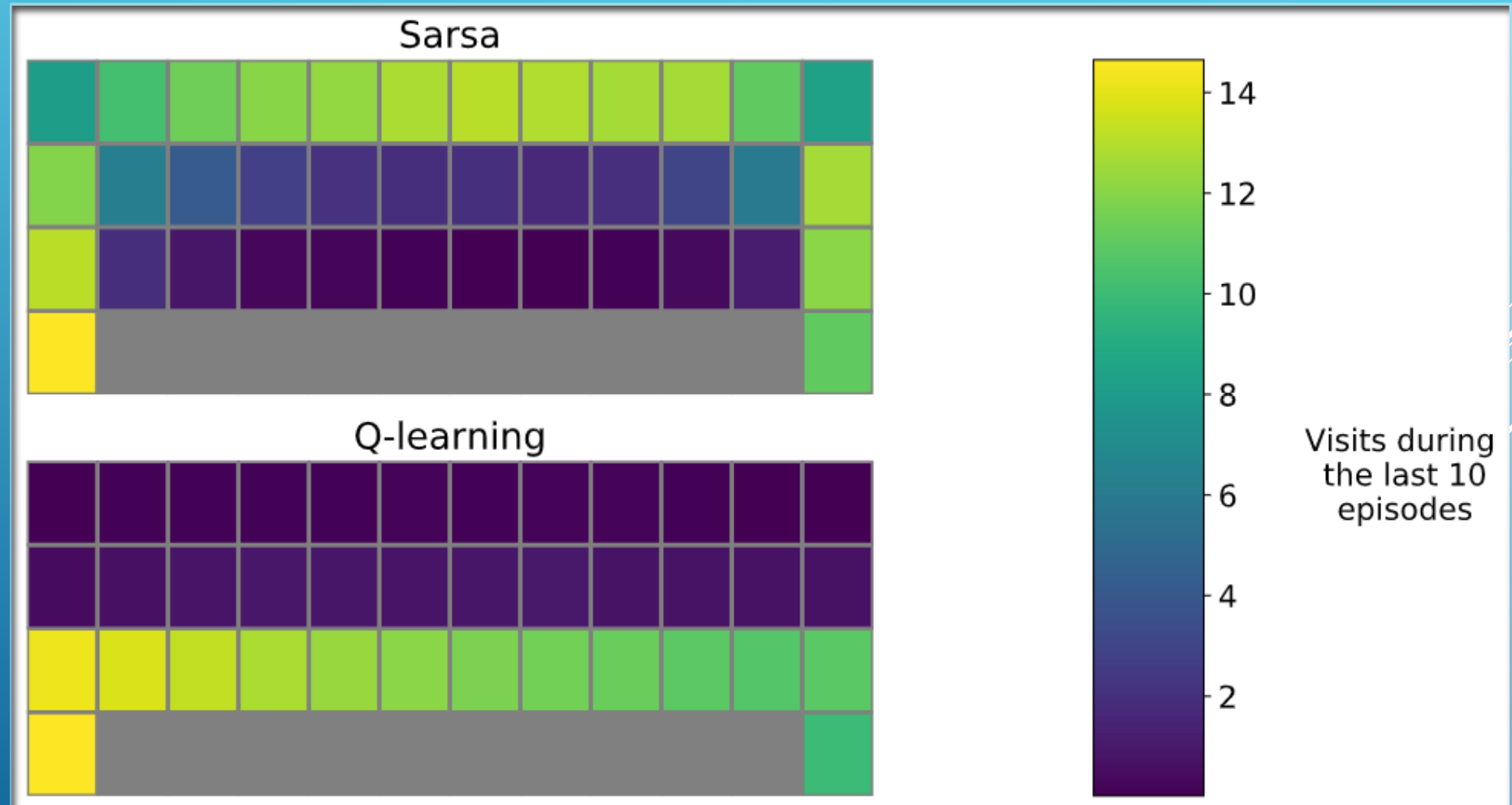
SARSA VS. Q-LEARNING REWARDS



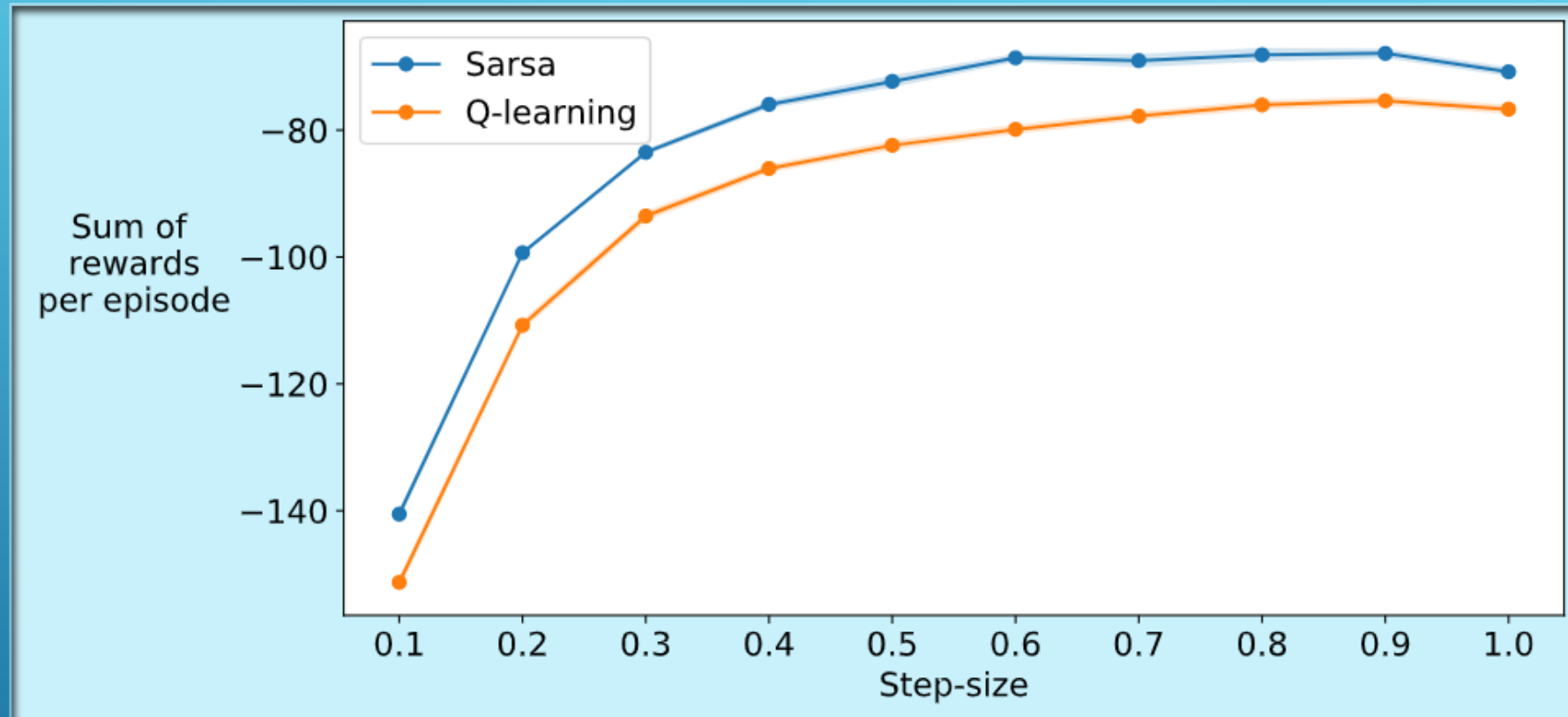
- 100 runs / 500 episodes per run.
- Average the sum of rewards for each episode over 100 runs.

Q-LEARNING VS. SARSA

- 100 runs / 500 episodes per run.
- Average the number of visits to a state during the last 10 episodes over 100 runs.



STEP-SIZE: SARSA VS. Q-LEARNING VS. EXPECTED SARSA



- 100 runs / 100 episodes per run.
- Average the sum of rewards for each episode over 100 runs for each step-size.

CONCLUSIONS: IMPLEMENTING REINFORCEMENT LEARNING APPLICATIONS

- The basic RL algorithms are a simple loop with incremental updates of the value functions and/or the policy and an internal model.
- An RL implementation must fill-out the details of the agent and the environment and keep these representations separate.
- A clean RL design:
 - Makes the interaction between an agent and its environment explicit.
 - Enables testing and experimentation by allowing different agents and environments to be substituted in a single framework.

CONCLUSIONS: SARSA VS. Q-LEARNING

- Q-Learning will learn the optimal policy for an MDP but cannot fully exploit it unless it stops exploring.
- If Q-Learning continues to explore, the total value per episode will be sub-optimal.
- The Sarsa algorithm can “play it safe” since it learns the policy it actually carries out.

FUTURE WORK: INVESTIGATE RL FRAMEWORKS

- OpenAI Gym
- Google Dopamine
- RLLib
- Keras-RL
- etc., etc.
- See Phil Winder of Winder Research:
 - <https://winderresearch.com/a-comparison-of-reinforcement-learning-frameworks-dopamine-rllib-keras-rl-coach-trfl-tensorforce-coach-and-more/>

DEVELOPING A SIMPLE REINFORCEMENT LEARNING APPLICATION USING RL-GLUE

Scott O'Hara

Metrowest Developers Machine Learning Group

09/02/2020