DEVELOPING A SIMPLE REINFORCEMENT LEARNING APPLICATION USING RL-GLUE

Scott O'Hara
Metrowest Developers Machine Learning Group
8/19/2020

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/

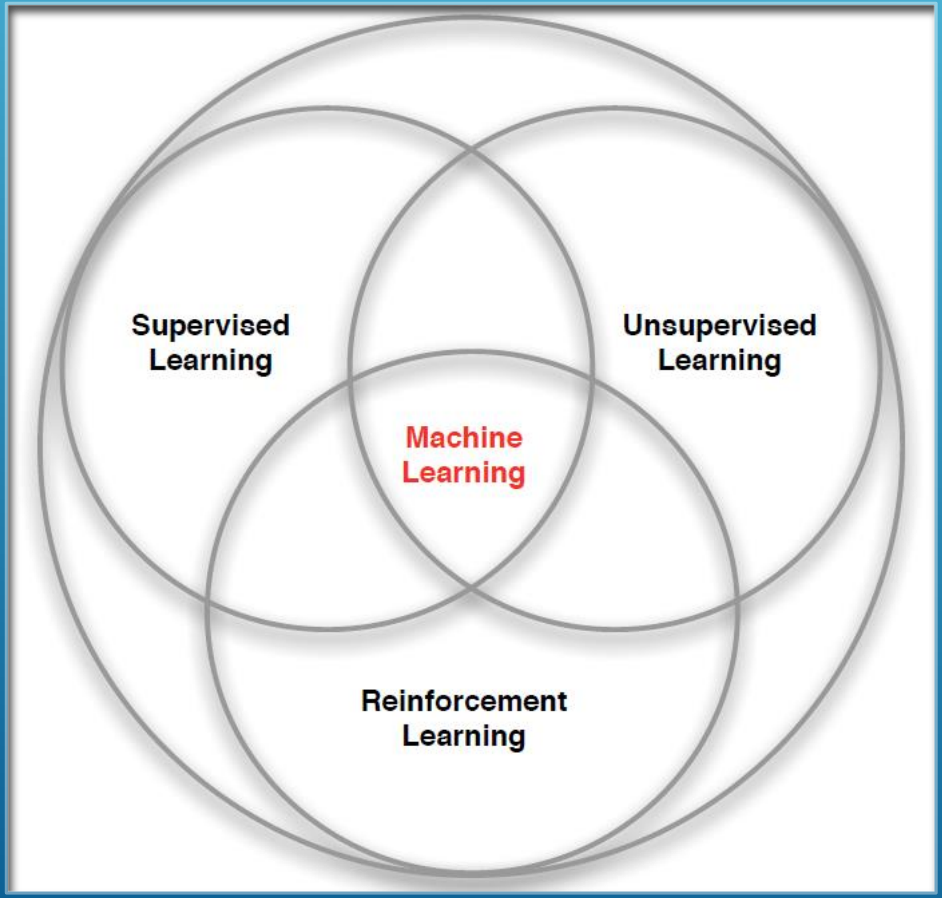
Reinforcement learning: An Introduction *R. S. Sutton and A. G. Barto, Second edition. Cambridge, Massachusetts: The MIT Press, 2018.*

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

- *https://www.davidsilver.uk/teaching/*

WHAT IS REINFORCEMENT LEARNING?

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



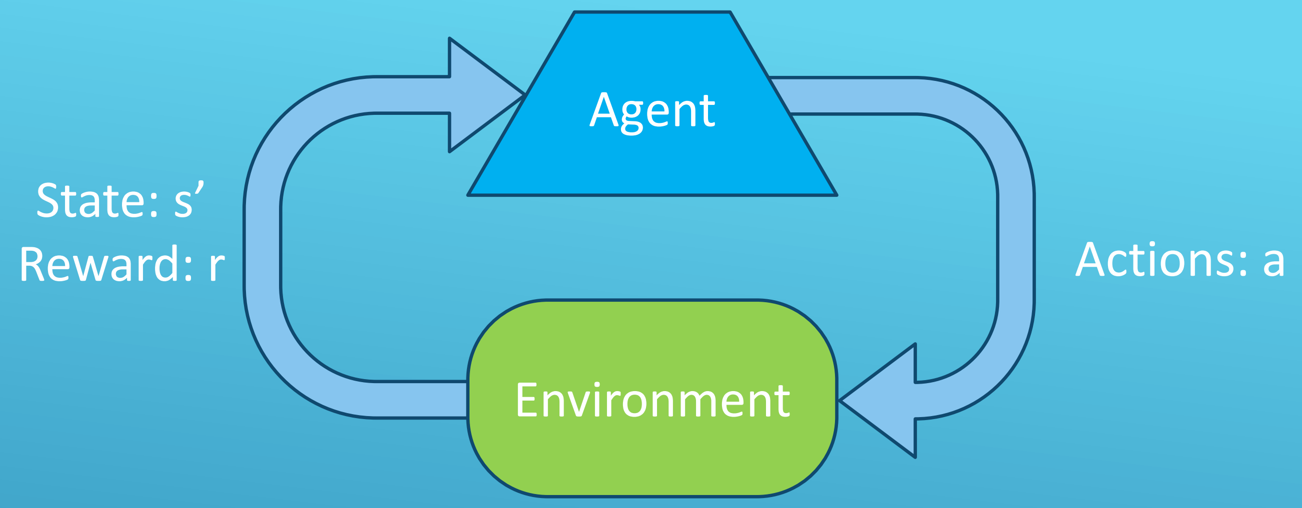
Supervised Learning – Learn a function from labeled data that maps input attributes to an output.

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

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

Credit: image borrowed from lecture slides David Silver, DeepMind, "Introduction to Reinforcement Learning

THE REINFORCEMENT LEARNING PROBLEM



- Agent must learn to act to maximize expected rewards.
- Agent knows the current state s, takes an action ***a***, receives a reward ***r*** and observes the next state ***s’***.

$$S_0, A_0, R_0, S_1, A_1, R_2, S_2, A_2, R_2, \ldots, S_n, A_n, R_n, S_T$$

- Agent has **no access** to the reward model ***r(s,a,s’)*** or the transition model ***T(s,a,s’)***.

MARKOV DECISION PROCESSES

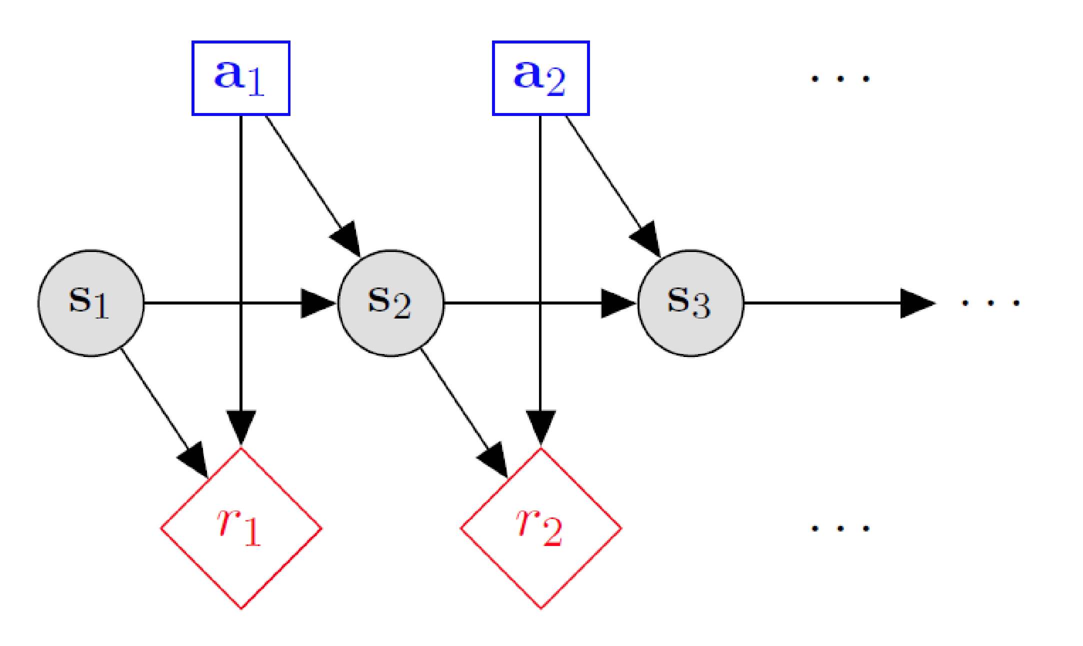
- **States:** $s_1, \ldots, s_n$
- **Actions:** $a_1, \ldots, a_m$
- **Reward model:**

$$\mathrm{R}(s, a, s') \in R$$

- **Transition model:**

$$T(s, a, s') = \mathrm{P}(s'|s, a)$$

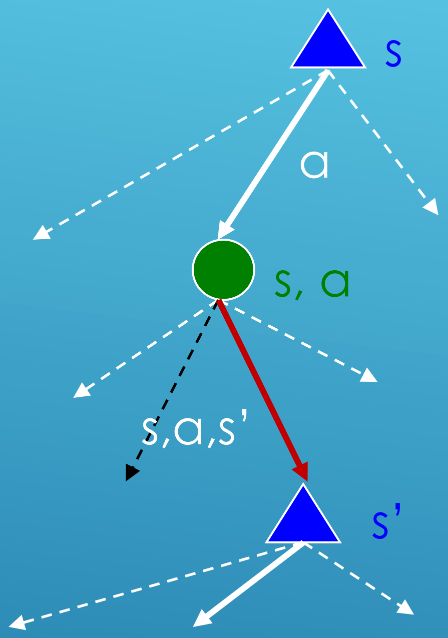
- **Discount factor:** $\gamma \in [0, 1]$



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

QUANTITIES TO OPTIMIZE

- The value (utility) of a **state s:**
  V(s) = expected utility starting in s and acting optimally thereafter.
- The value (utility) of a **q-state (s,a):**
  Q(s,a) = expected utility when taking action a from state s and acting optimally thereafter.
- The policy π:
  π(a|s) = probability of action a from state s



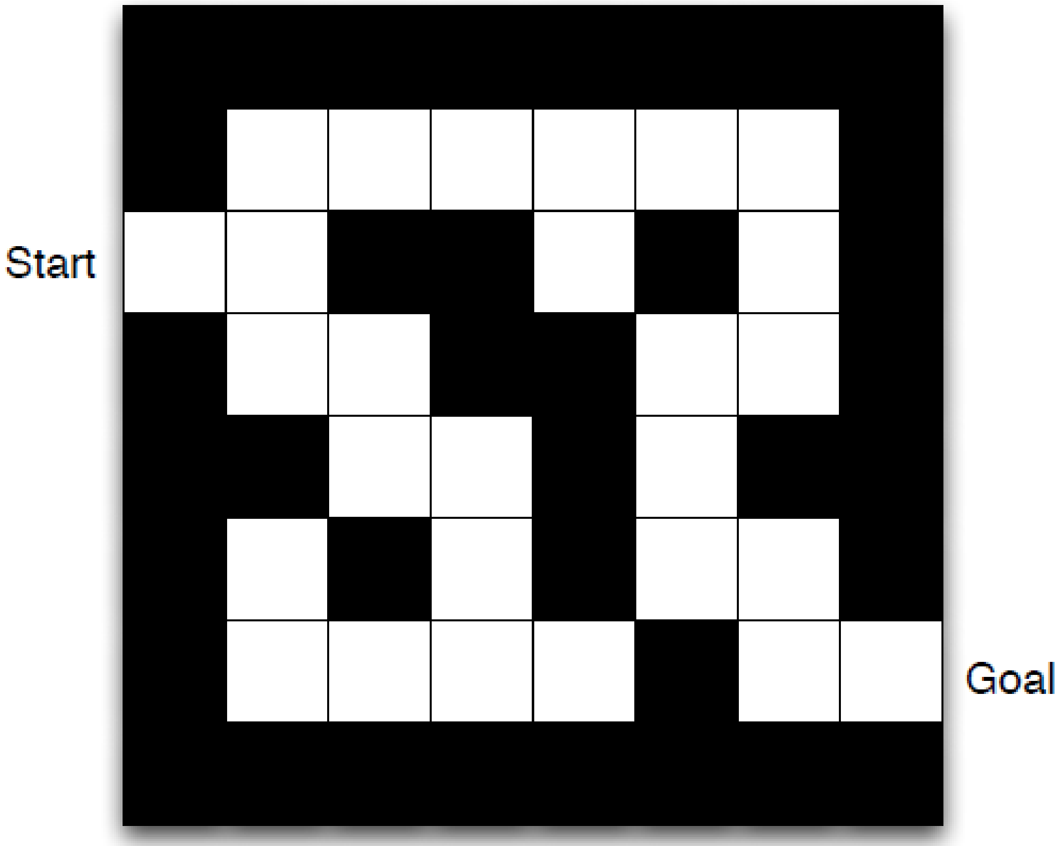
s is a *state*

(s, a) is a *q-state*

(s,a,s') is a *transition*

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

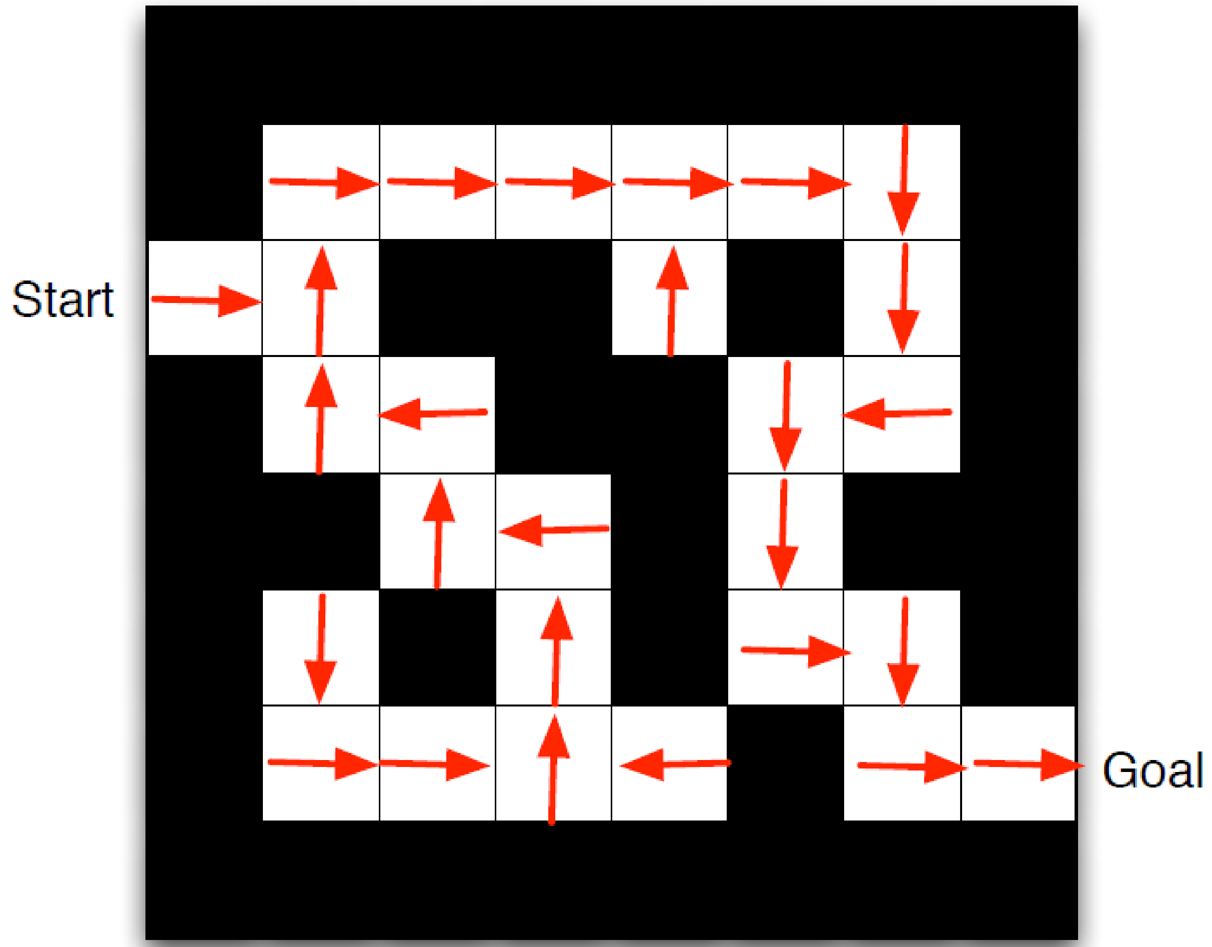
Maze Example



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

Credit: Reinforcement Learning, *David Silver, University College London / Google DeepMind, 2015.*

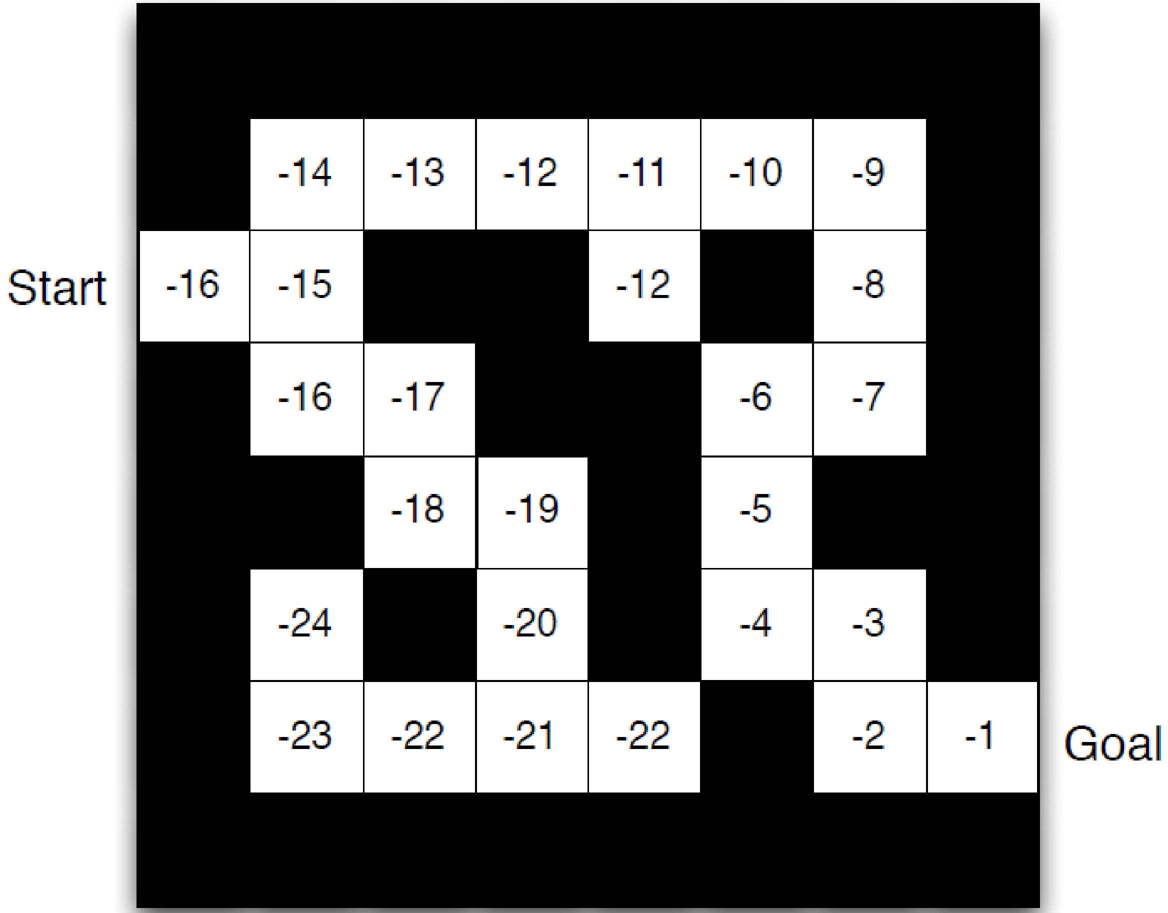
Maze Example: **Policy**



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

Credit: Reinforcement Learning, *David Silver, University College London / Google DeepMind, 2015.*

Maze Example: **Value Function**



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

Credit: Reinforcement Learning, *David Silver, University College London / Google DeepMind, 2015.*

TEMPORAL DIFFERENCE (TD) LEARNING?

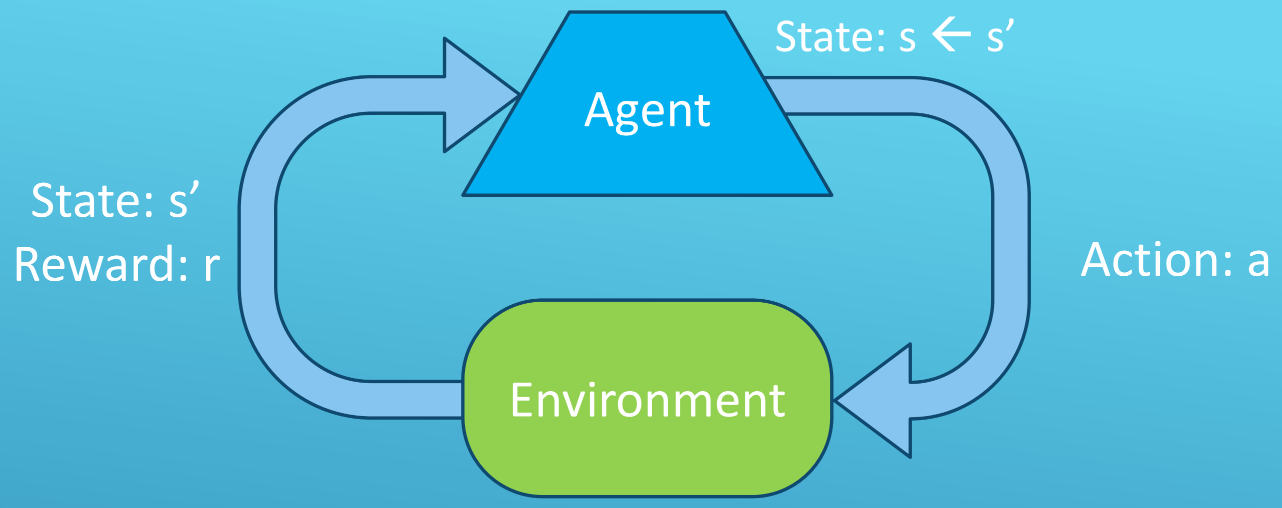
- TD-Learning is a kind of *prediction learning* that takes advantage of the temporal structure of learning to predict.

- In prediction 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.

- Sarsa and Q-Learning are examples of Temporal Difference Learning algorithms.

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

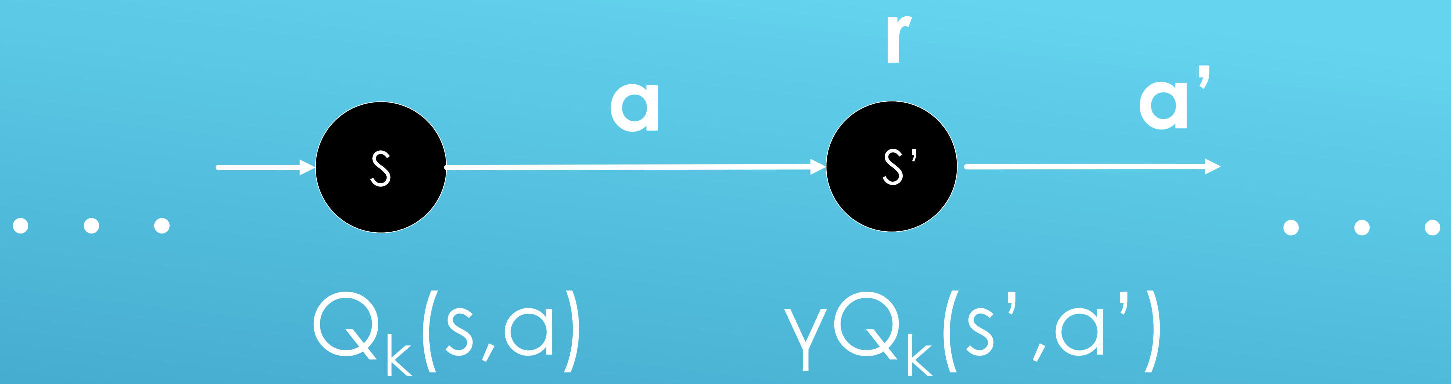
THE REINFORCEMENT LEARNING PROBLEM



Agent knows the current state s, takes an action ***a***, receives a reward ***r*** and observes the next state ***s’***.

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

SARSA Update Rule



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

The Future
(The Target)

The Present
(The Prediction)

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

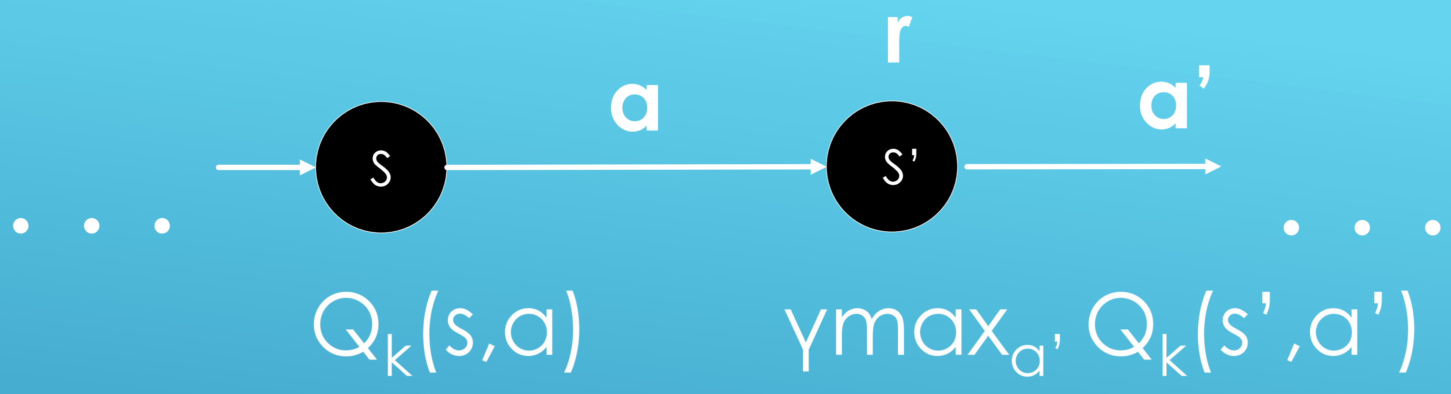
SARSA Algorithm

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

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)
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
        $S \leftarrow S'$; $A \leftarrow A'$;
    until S is terminal

Q-Learning Update Rule



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

The Future
(The Target)

The Present
(The Prediction)

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

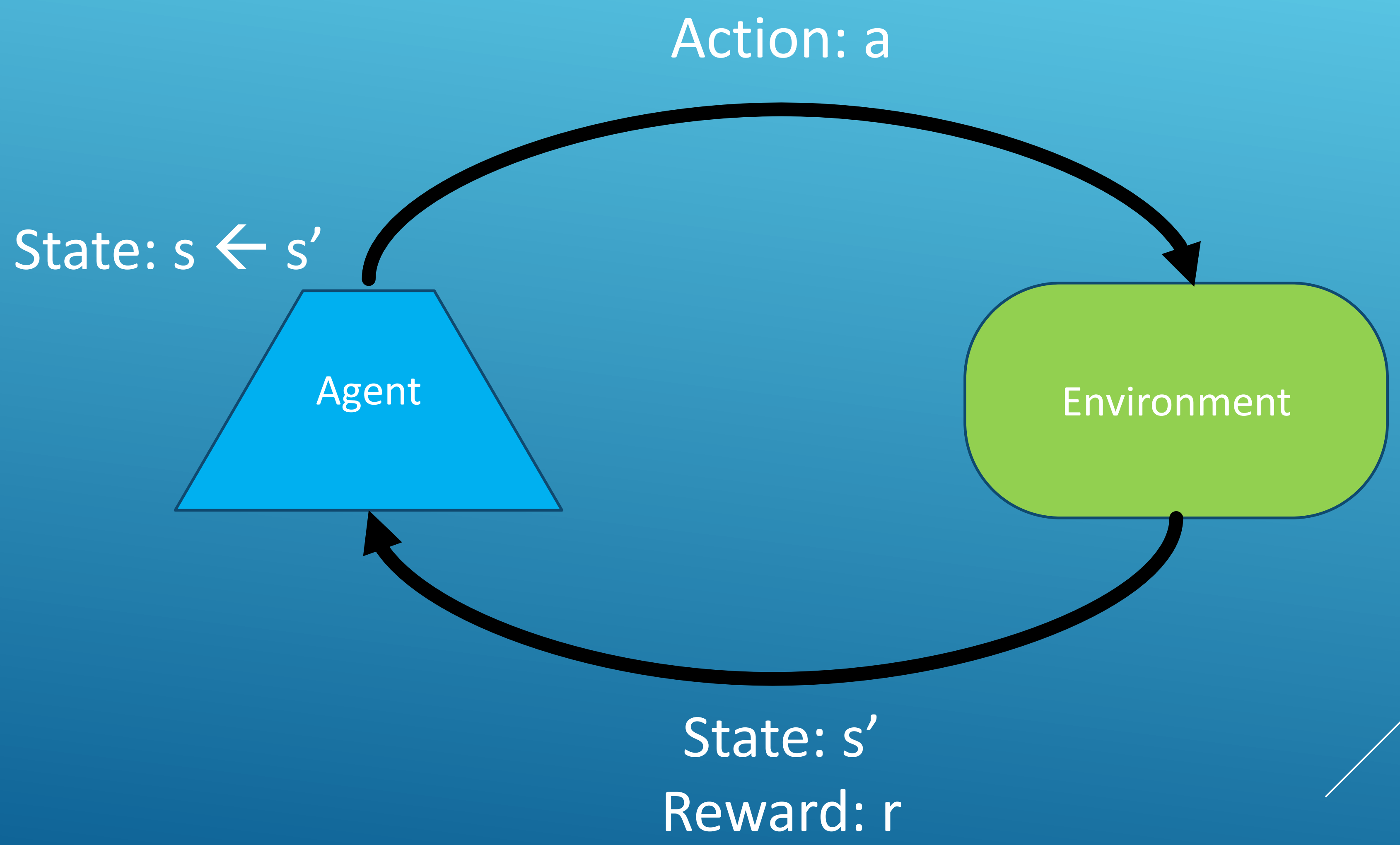
Q-learning Algorithm

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

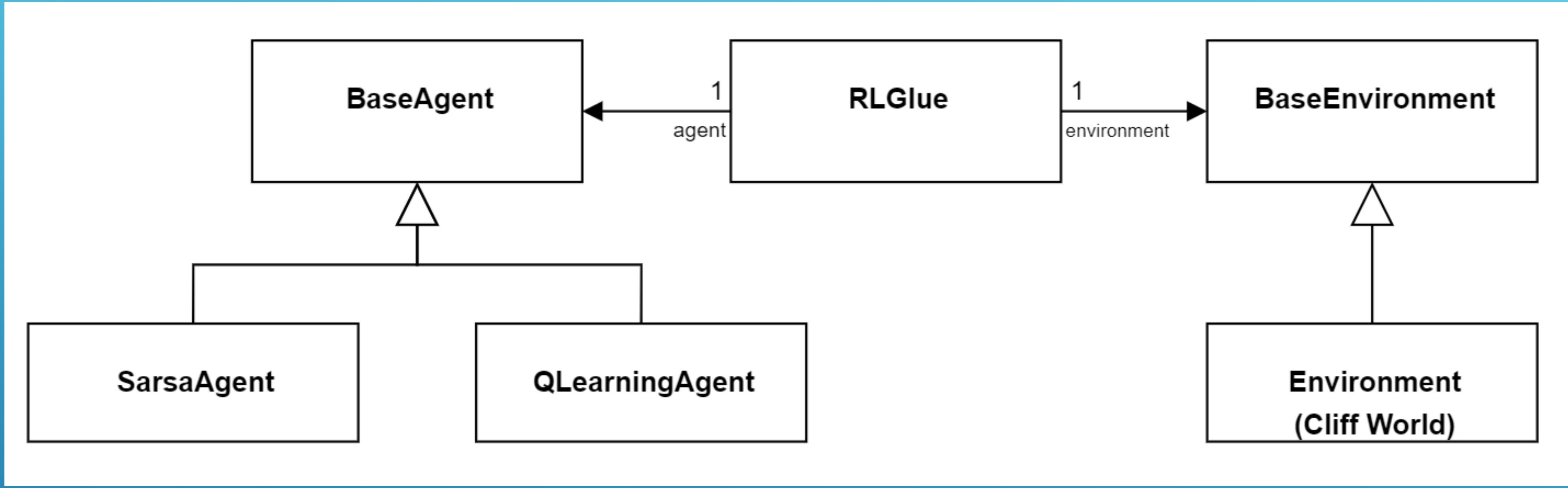
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

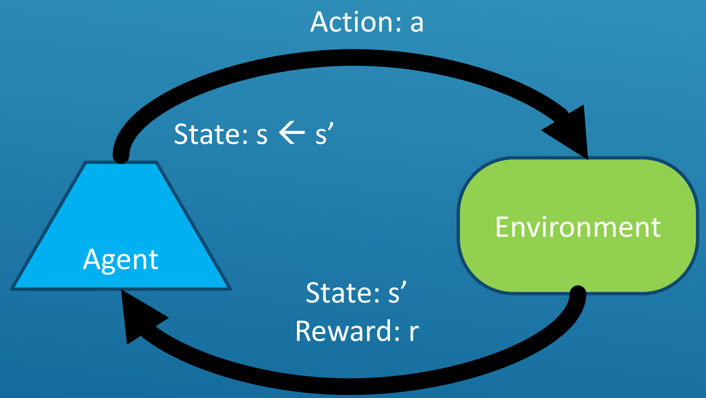
Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\left[R + \gamma \max_a Q(S', a) - Q(S, A)\right]$
    $S \leftarrow S'$
  until S is terminal

The Reinforcement Learning Loop



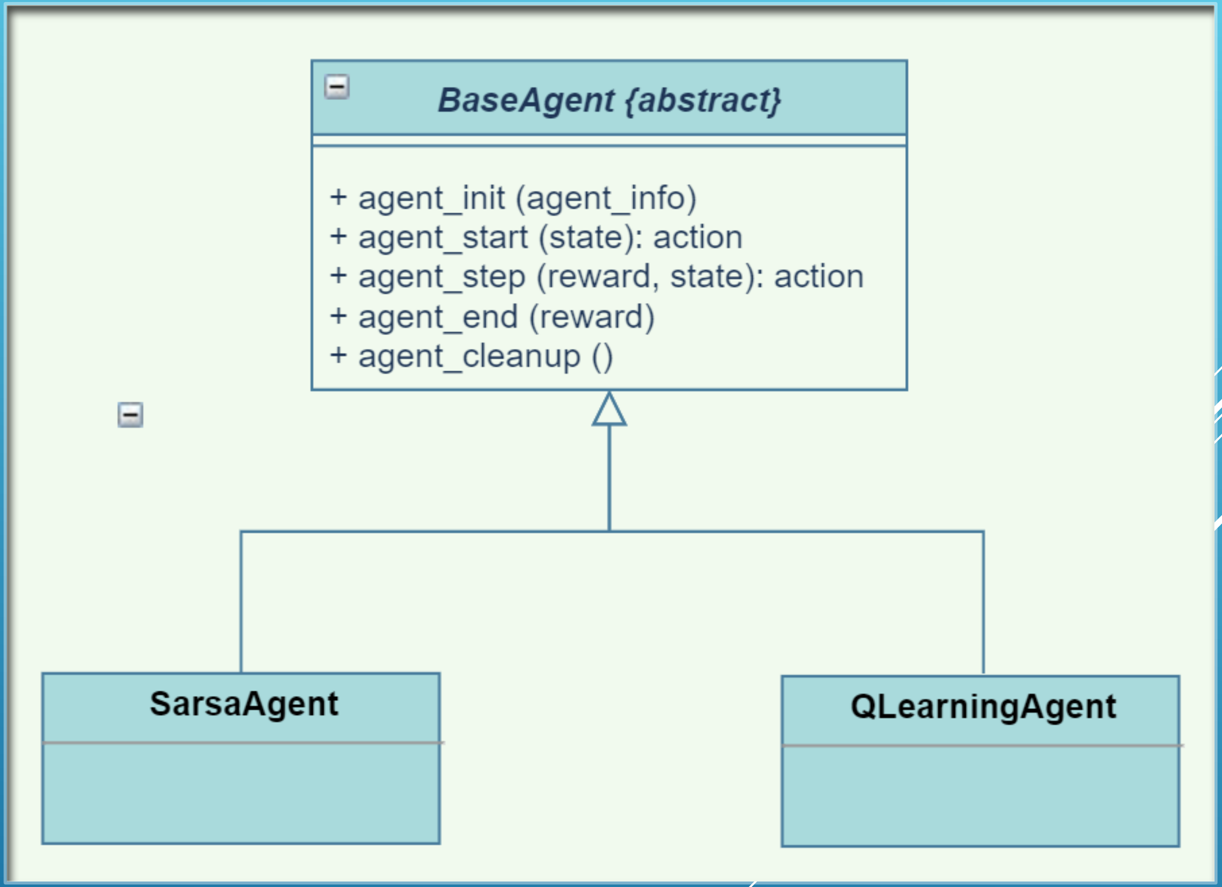
Implementing the RL Loop with RLGlue





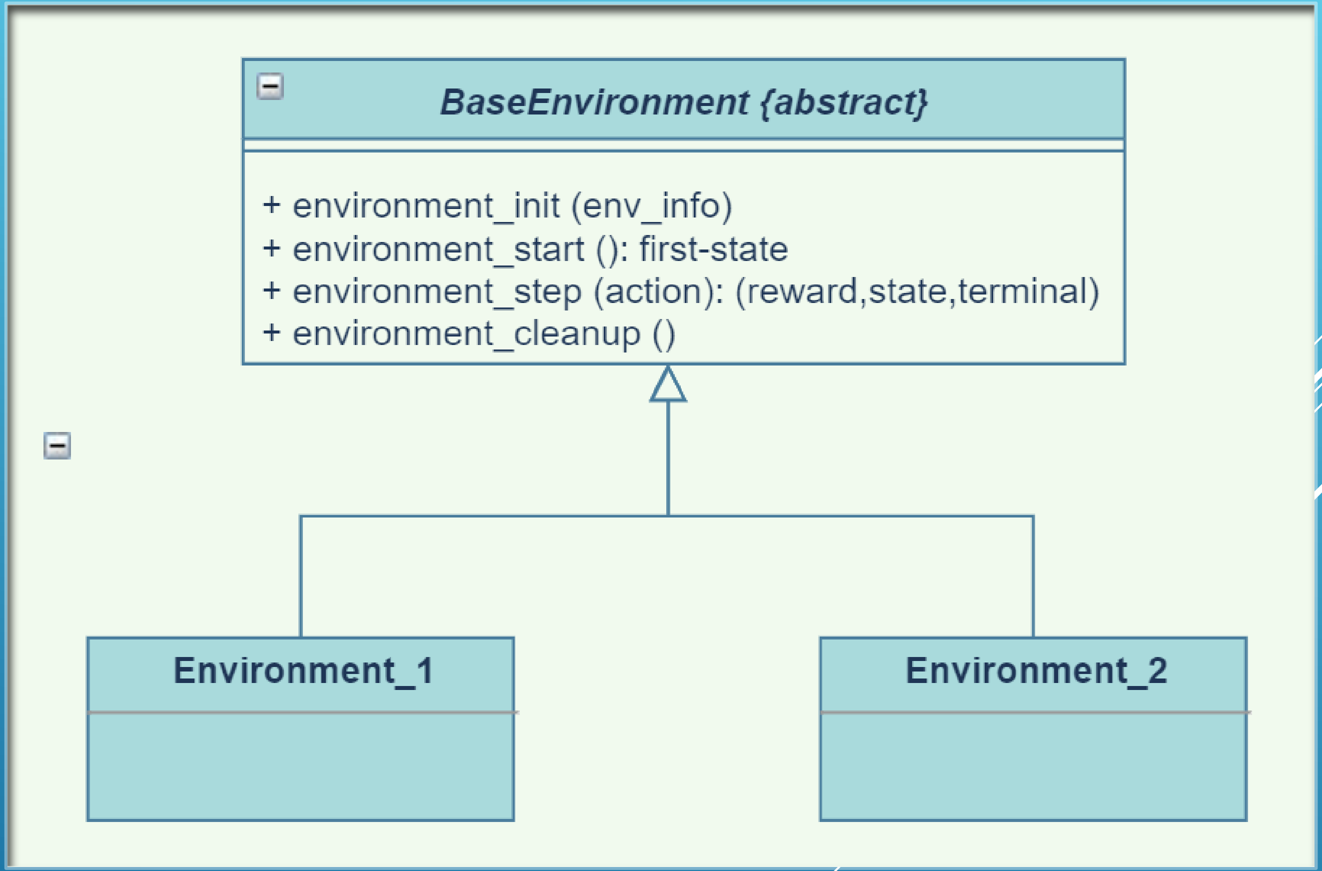
THE BASE AGENT CLASS

- **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

+ __init__(Agent, Environment)
+ rl_init (env_info)
+ rl_start (): first-state
+ rl_step (): (reward,state,action,terminal)
+ rl_episode()
+ rl_cleanup ()

- **__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__()

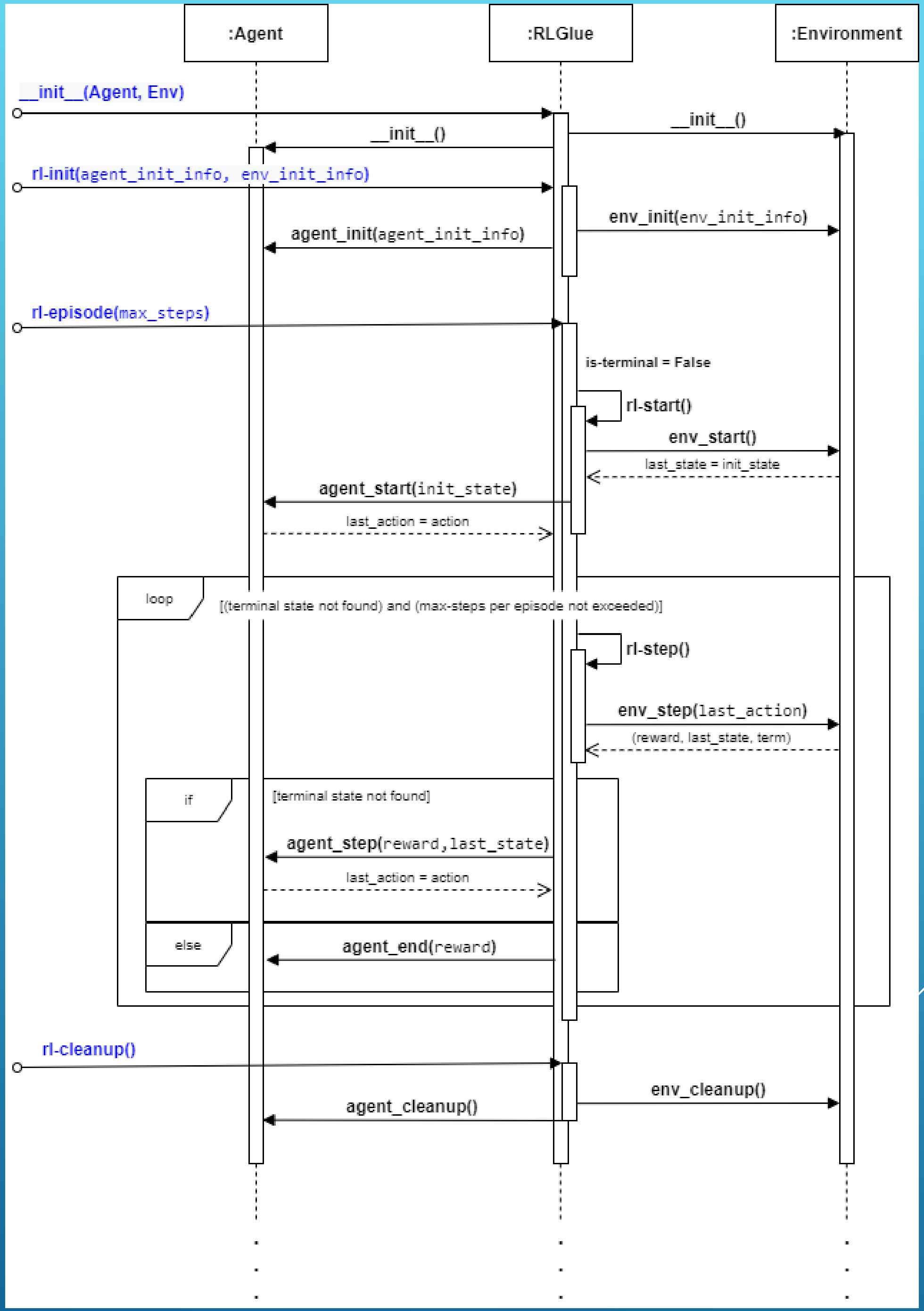
rl-init()

rl-episode()*

rl-start()

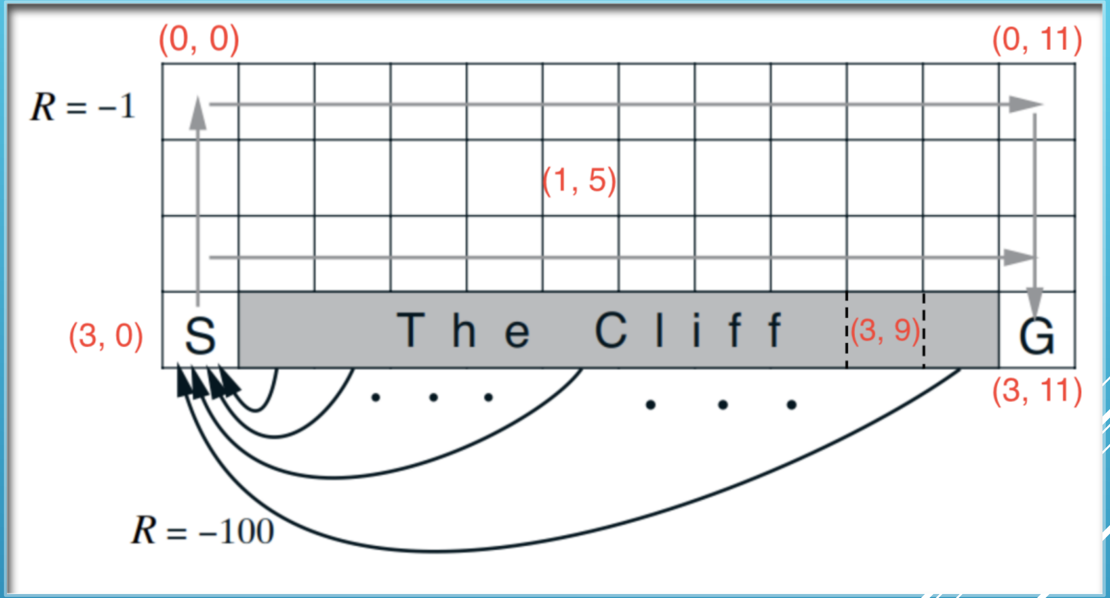
rl-step()*

rl-cleanup()



RL ENVIRONMENT: CLIFFWORLD

- **States: <x, y>** 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.)

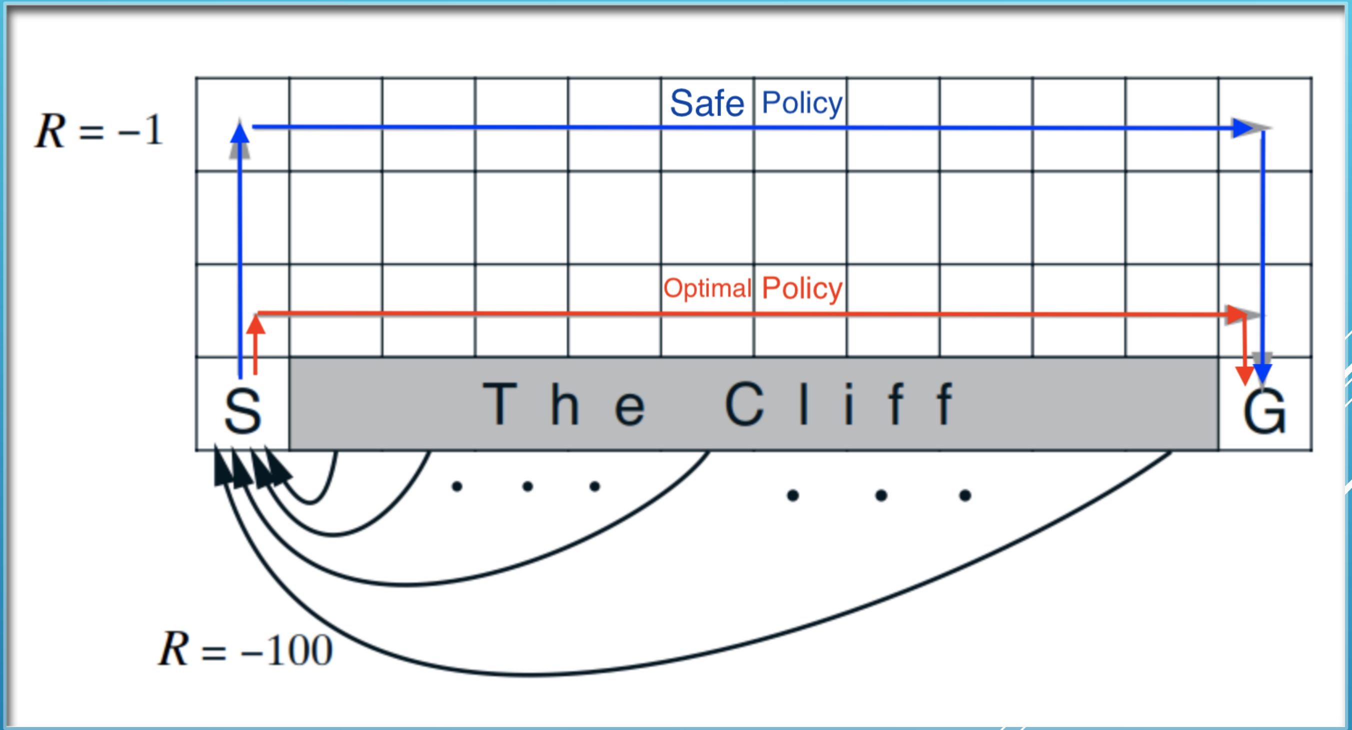


- **Discount factor: γ = 1.0**

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

OPTIMAL VS. "SAFE" POLICIES: Q-LEARNING

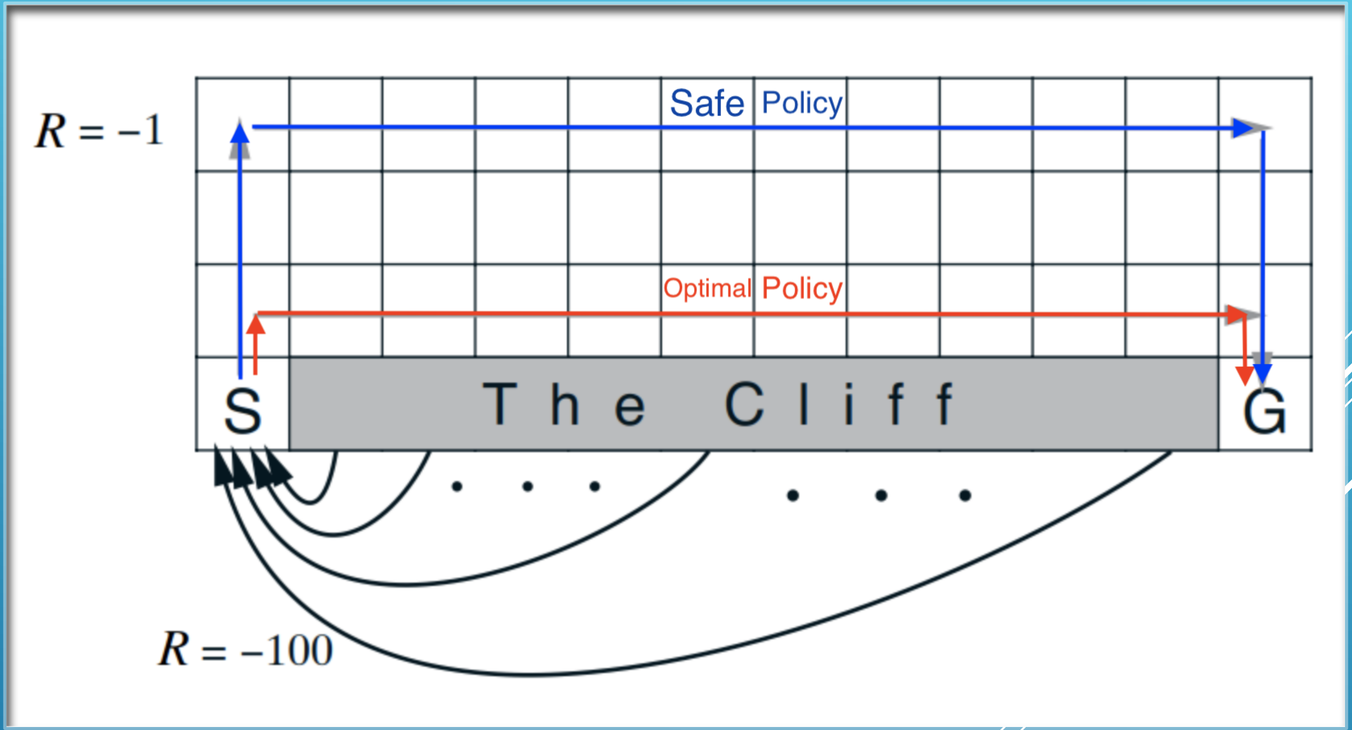
- Q-learning learns the optimal policy.
- However, q-learning must stop exploring and change to complete exploitation mode to take advantage of this.
- If q-learning continues to explore (off-policy), it will often get bad results since exploration will lead it to step over the cliff.



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

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.

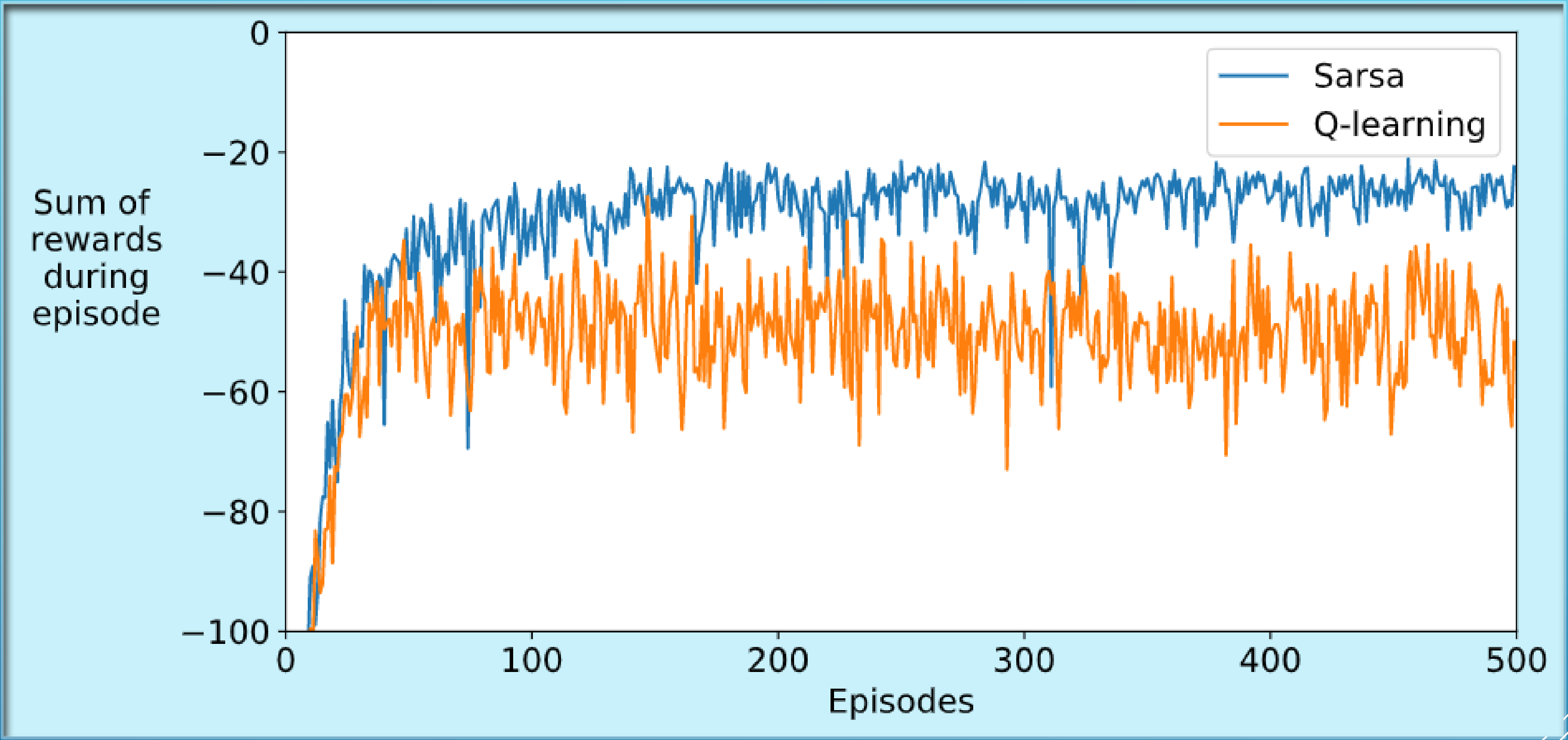


Credit: Sample-based Learning Methods *(M. White and A. White), University of Alberta, Alberta Machine Intelligence Institute, Coursera.*

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.

SARSA VS. Q-LEARNING REWARDS

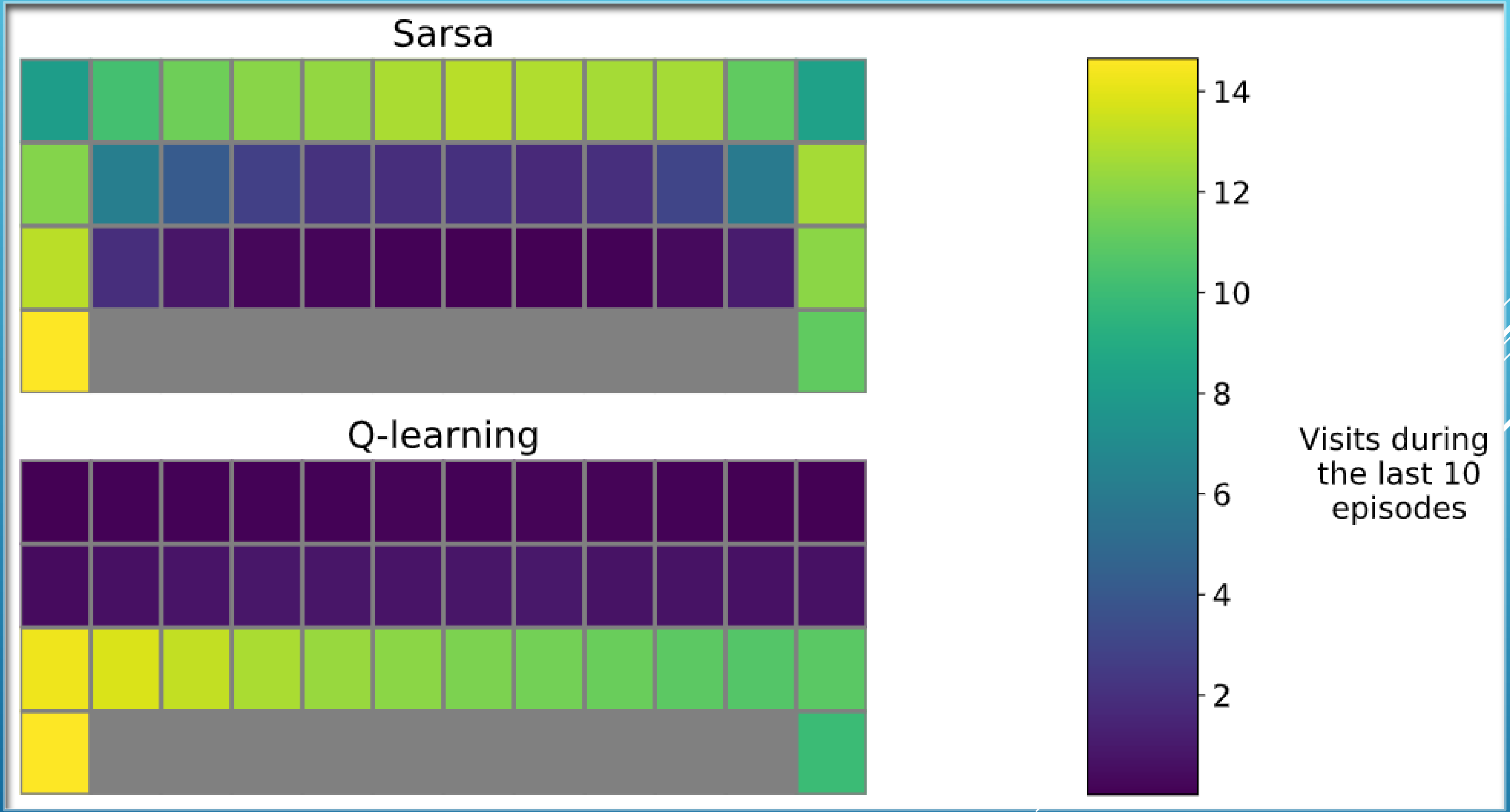


- 100 runs / 500 episodes per run.
- Average the sum of rewards for each episode over 100 runs.

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

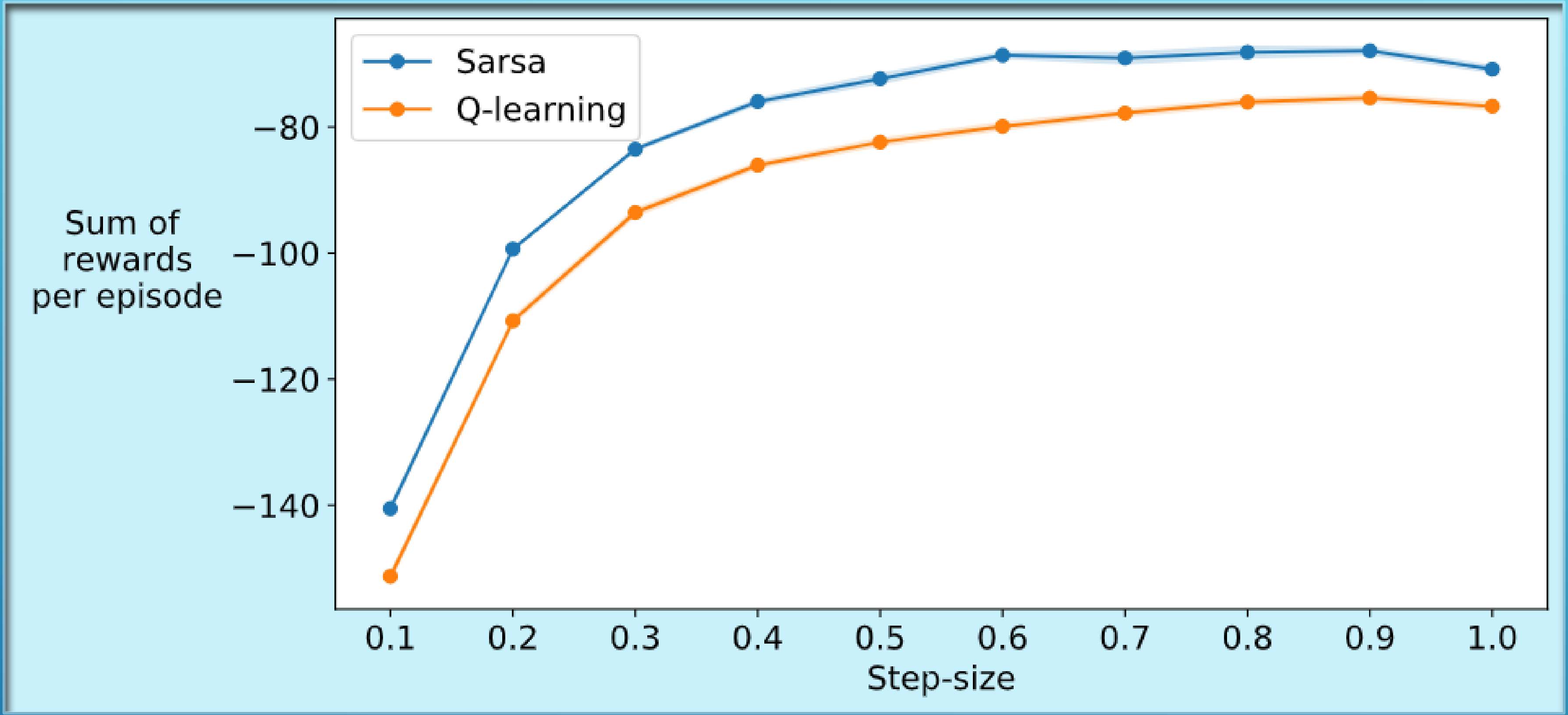
Q-LEARNING VS. SARSA

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



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

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.

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

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
8/19/2020