A Review of RL Algorithms and Its Application in UAV-BS Deployment

Linh T. Hoang

Jan 17, 2023

Contents

Part I: A Review of RL Algorithms

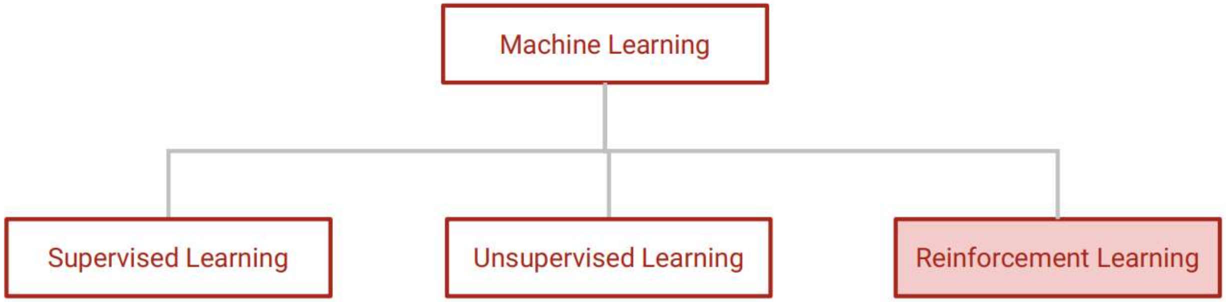
- Reinforcement Learning: The Basics
- A Brief Intro to Deep Reinforcement Learning

Part II: RL for UAV-BS Deployment

- Deployment of UAV-mounted Base Stations
- Some Initial Results
- The Road Ahead

Part I: A Review of RL Algorithms

A Review of Machine Learning Paradigms



Supervised Learning

Training with Labeled Data

Application: Image Classification, Speech Recognition, Regression, etc.

Unsupervised Learning

Find hidden features of Unlabeled Data

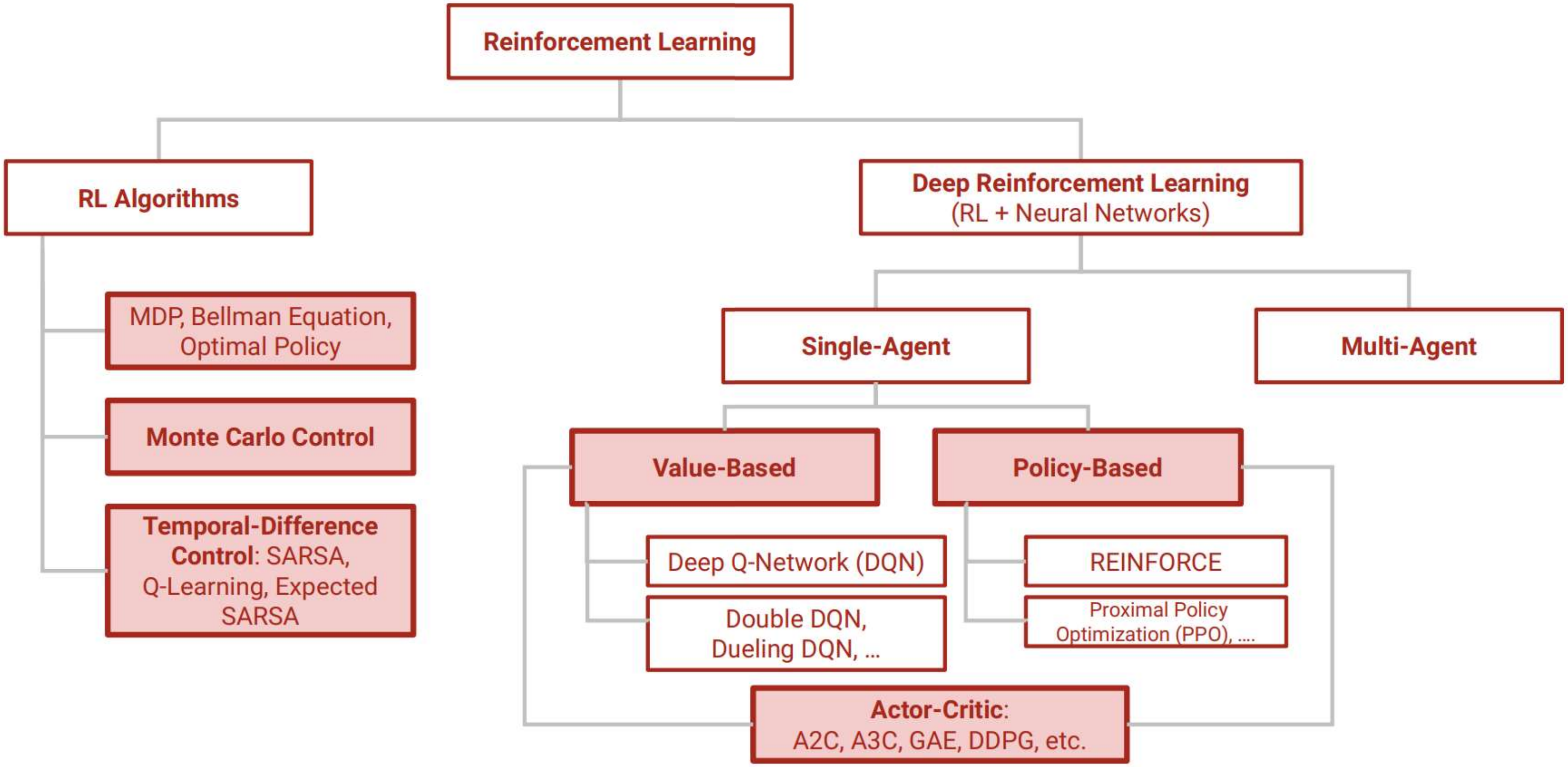
Application: Clustering, Anomaly Detection, Dimensionality Reduction, etc.

Reinforcement Learning

Training with no data beforehand but through interactions with the environment

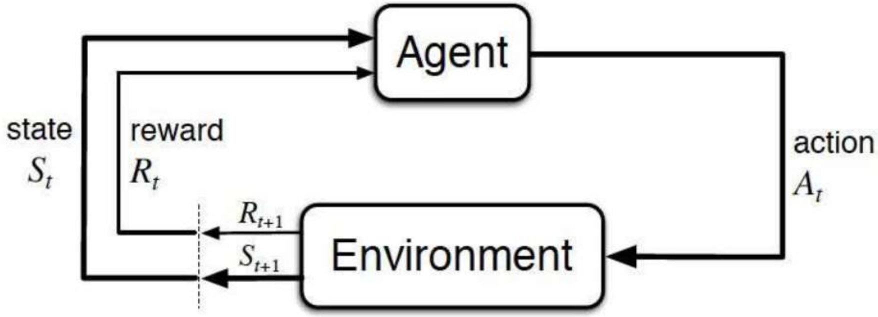
Application: Decision making, Robot Navigation, Game AI, etc.

A Taxonomy of RL Algorithms



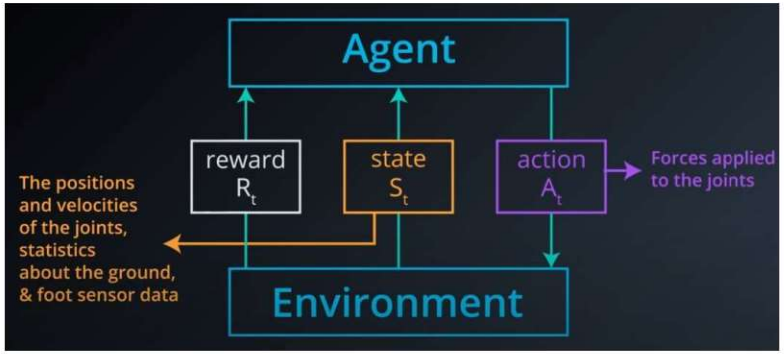
RL Formulation

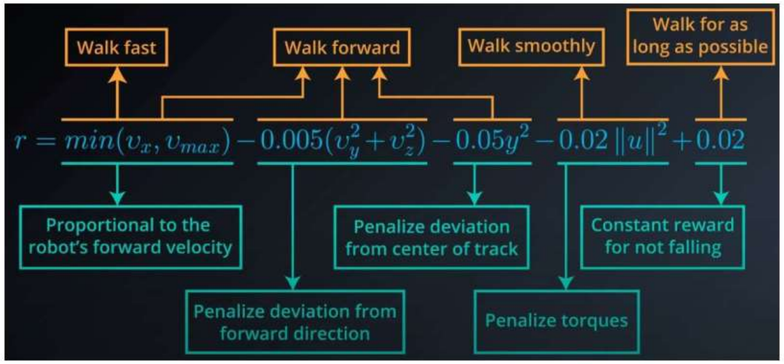
- An **agent** learning to interact with its **environment**.
- At each time step, the agent receives the environment's **state**, and the agent must choose an appropriate **action** in response.
- One time step later, the agent receives a **reward** (the environment indicates whether the agent has responded appropriately to the state) and a new state.
- The agent aim to maximize the **expected cumulative reward** (i.e., the expected sum of rewards attained over all time steps).

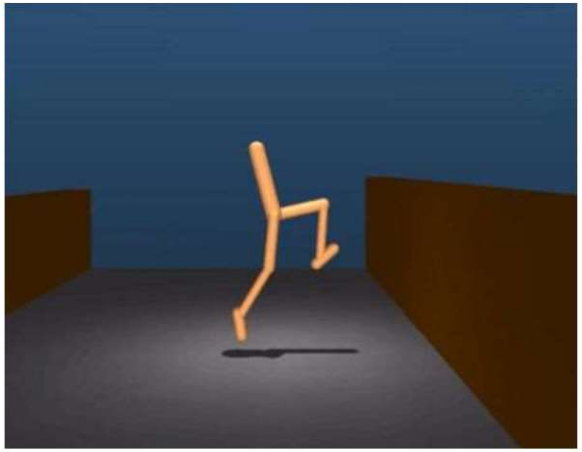


The agent-environment interaction in reinforcement learning.
(Sutton and Barto, 2017)

Example: An RL agent learn how to walk (1/2)







https://deepmind.google/discover/blog/producing-flexible-behaviours-in-simulated-environments/

(Google DeepMind) Emergence of Locomotion Behaviours in Rich Environments
https://www.youtube.com/watch?v=hx_bgoTF7bs

Example: An RL agent learn how to walk (2/2)

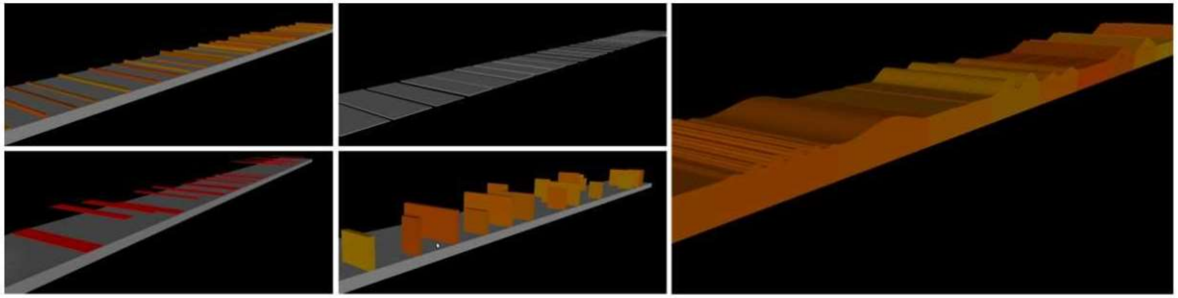


Figure 3: Examples of the terrain types used in the experiments. Left to right and top to bottom: *hurdles*, *platforms*, *gaps*, *slalom walls*, *variable terrain*.

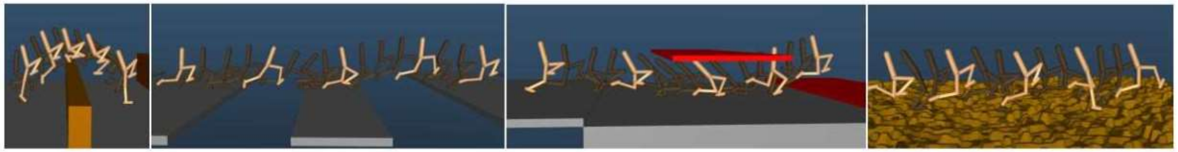


Figure 4: *Walker skills*: Time-lapse images of a representative *Planar Walker* policy traversing rubble; jumping over a hurdle; jumping over gaps and crouching to pass underneath a platform.

N. Heess et al., “Emergence of Locomotion Behaviours in Rich Environments.” arXiv, 2017. doi: 10.48550/ARXIV.1707.02286.

RL Formulation using Markov Decision Process

A **(finite) Markov Decision Process (MDP)** is defined by:

- a (finite) set of states $\mathcal{S}$ (or $\mathcal{S}^+$, in the case of an episodic task)
- a (finite) set of actions $\mathcal{A}$
- a set of rewards $\mathcal{R}$
- the one-step dynamics of the environment
- the discount rate $\gamma \in [0, 1]$

At an arbitrary time step t, the agent-environment interaction has evolved as a sequence of states, actions, and rewards

$$\boxed{(S_0, A_0, R_1, S_1, A_1, \ldots, R_{t-1}, S_{t-1}, A_{t-1}, R_t, S_t, A_t).}$$

When the environment responds to the agent at time step $t+1$, it considers only the state and action at the previous time step (S_t, A_t).

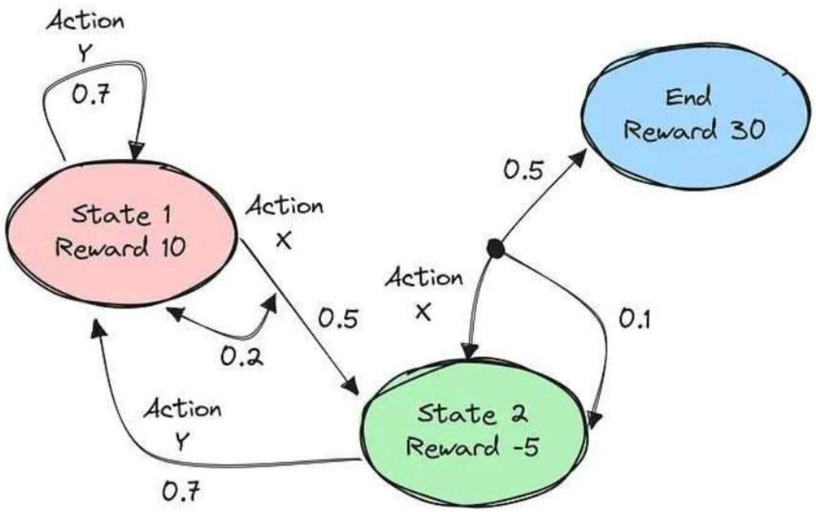
$$\boxed{p(s', r|s, a) \doteq \mathbb{P}(S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a)}$$

for each possible $s', r, s, \text{and } a$. These conditional probabilities are said to specify the **one-step dynamics** of the environment.

The return (the cumulative reward) at time step *t*:

$$\boxed{G_t := R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots.}$$

In each time step, the agent select an action with a goal of maximizing the **expected (discounted)** return.



An example of MDP for RL formulation

https://python.plainenglish.io/understanding-markov-decision-processes-17e852cd9981

State-Value Function and Bellman Equation

State-Value Functions

- The **state-value function** for a policy π is denoted v_π. For each state $s \in \mathcal{S}$, it yields the expected return if the agent starts in state s and then uses the policy to choose its actions for all time steps. That is, $\boxed{v_\pi(s) \doteq \mathbb{E}_\pi[G_t|S_t = s]}$. We refer to $v_\pi(s)$ as the **value of state s under policy π**.

The discounted return (cumulative reward) at time t:

$$G_t := R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots.$$

Bellman Equations

- The **Bellman expectation equation for v_π** is: $\boxed{v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s]}$.

Optimality

- A policy π' is defined to be better than or equal to a policy π if and only if $v_{\pi'}(s) \geq v_\pi(s)$ for all $s \in \mathcal{S}$.

Action-Value Function and Optimal Policies

Action-Value Functions

- The **action-value function** for a policy π is denoted q_π. For each state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$, it yields the expected return if the agent starts in state s, takes action a, and then follows the policy for all future time steps. That is, $\boxed{q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t|S_t=s, A_t=a]}$. We refer to $q_\pi(s,a)$ as the **value of taking action a in state s under a policy π** (or alternatively as the **value of the state-action pair s, a**).
- All optimal policies have the same action-value function q_*, called the **optimal action-value function**.

The discounted return (cumulative reward) at time *t*:

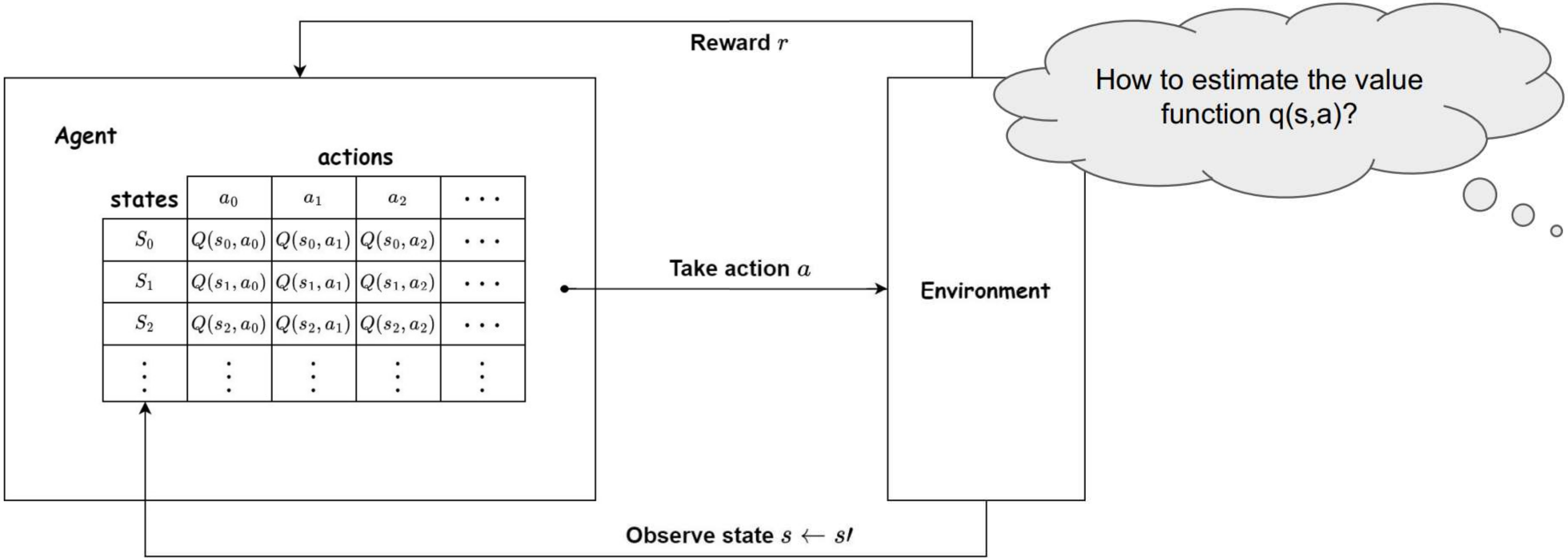
$$G_t := R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots.$$

Optimal Policies

- Once the agent determines the optimal action-value function q_*, it can quickly obtain an optimal policy π_* by setting $\boxed{\pi_*(s) = \arg\max_{a \in \mathcal{A}(s)} q_*(s,a)}$.

The problem now is how to estimate the optimal value function q*(s,a)

Q-Table for the action-value function $q_\pi(s, a)$



https://wikidocs.net/174536

RL Solution: Monte Carlo Control

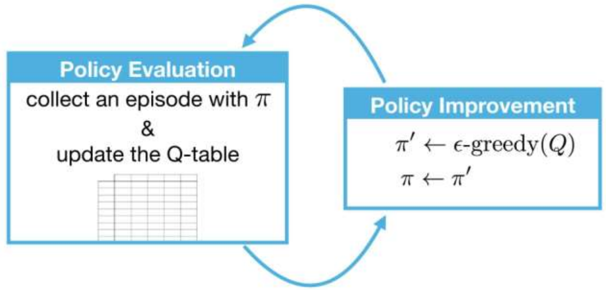
How to estimate the value function q(s,a)?

MC Control alternates between policy evaluation and policy improvement steps to recover the optimal policy π∗.

Algorithm 11: First-Visit Constant-α (GLIE) MC Control

Input: positive integer *num_episodes*, small positive fraction α, GLIE $\{\epsilon_i\}$
Output: policy π ($\approx \pi_*$ if *num_episodes* is large enough)
Initialize Q arbitrarily (e.g., $Q(s,a) = 0$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$)
for $i \leftarrow 1$ ***to*** *num_episodes* **do**
  $\epsilon \leftarrow \epsilon_i$
  $\pi \leftarrow \epsilon\text{-greedy}(Q)$
  Generate an episode $S_0, A_0, R_1, \ldots, S_T$ using π
  for $t \leftarrow 0$ ***to*** $T-1$ **do**
    if *(S_t, A_t) is a first visit (with return G_t)* **then**
      $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(G_t - Q(S_t, A_t))$
  end
end
return π

$$Q(S_t, A_t) \leftarrow (1-\alpha)Q(S_t, A_t) + \alpha G_t$$



Monte Carlo Control

RL Solution: Temporal Difference Control

How to estimate the value function q(s,a)?

Monte Carlo (MC) control methods require an agent to complete an entire episode of interaction before updating the Q-table.

Temporal Difference (TD) methods will instead update the Q-table <u>after every time step</u>.

Algorithm 14: Sarsamax (Q-Learning)

Input: policy π, positive integer *num_episodes*, small positive fraction α, GLIE $\{\epsilon_i\}$
Output: value function Q ($\approx q_\pi$ if *num_episodes* is large enough)
Initialize Q arbitrarily (e.g., $Q(s,a) = 0$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$, and $Q(\textit{terminal-state}, \cdot) = 0$)
for $i \leftarrow 1$ ***to*** *num_episodes* **do**
    $\epsilon \leftarrow \epsilon_i$
    Observe S_0
    $t \leftarrow 0$
    repeat
        Choose action A_t using policy derived from Q (e.g., ϵ-greedy)
        Take action A_t and observe R_{t+1}, S_{t+1}
        $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t))$
        $t \leftarrow t + 1$
    until S_t *is terminal*;
end
return Q

$$Q(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}}$$

(From Monte Carlo Control)

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(\underbrace{G_t}_{\text{alternative estimate}} - \underbrace{Q(S_t, A_t)}_{\text{current estimate}})$$

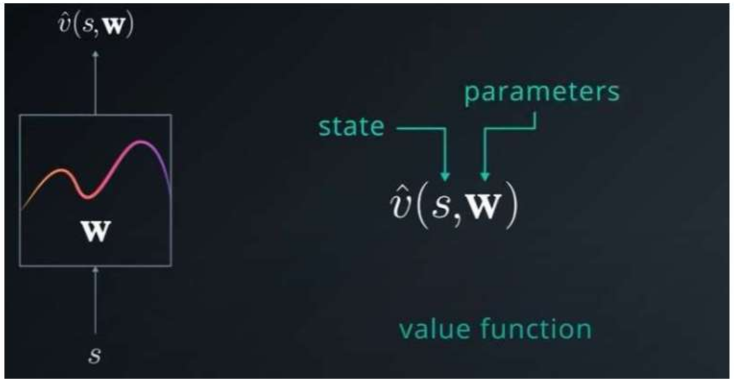
(From Temporal-Difference Control)

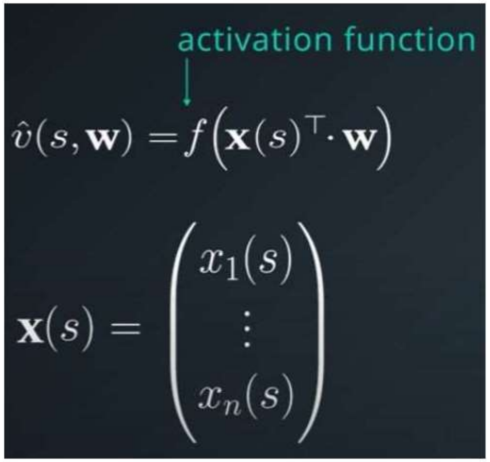
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(\underbrace{R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})}_{\text{alternative estimate}} - \underbrace{Q(S_t, A_t)}_{\text{current estimate}})$$

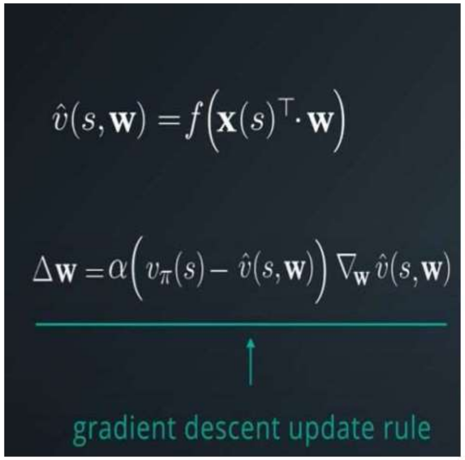
TD Control: Sarsa Algorithm

RL in Continuous Space

Instead of using a Q-Table, we can adopt Deep Neural Networks (DNN) as Nonlinear Function Approximation for the value functions.

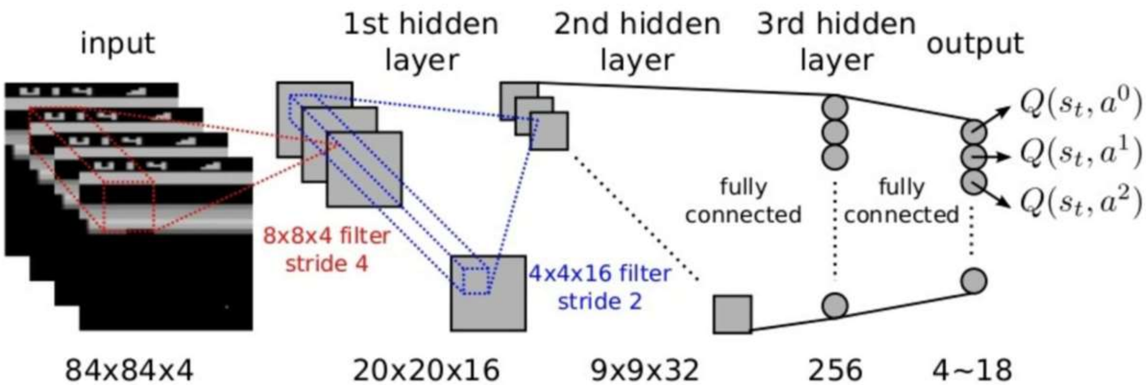






Deep Reinforcement Learning: An Example

Reinforcement Learning (RL): use **Q-tables** as an estimate of the value functions.
Deep RL: adopt **deep neural networks (DNN)** to estimate the value functions or directly form up a policy.



Example: Google DeepMind's Deep Q-Network (DQN) for playing Atari games [1].

Demon Attack

[1] V. Mnih et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

Training the neural network in Deep Q-Network [1]

$$J(\mathbf{w}) = \mathbb{E}_\pi\left[\left(q_\pi(S,A) - \hat{q}(S,A,\mathbf{w})\right)^2\right]$$

$$\nabla_\mathbf{w} J(\mathbf{w}) = -2\left(q_\pi(S,A) - \hat{q}(S,A,\mathbf{w})\right)\nabla_\mathbf{w}\hat{q}(S,A,\mathbf{w})$$

$$\begin{aligned}\Delta\mathbf{w} &= -\alpha\frac{1}{2}\nabla_\mathbf{w} J(\mathbf{w})\\ &= \alpha\left(q_\pi(S,A) - \hat{q}(S,A,\mathbf{w})\right)\nabla_\mathbf{w}\hat{q}(S,A,\mathbf{w})\end{aligned}$$

$$\Delta\mathbf{w} = \alpha\Big(\underbrace{\underbrace{R + \gamma\max_a \hat{q}(S',a,\mathbf{w})}_{\text{TD target}} - \underbrace{\hat{q}(S,A,\mathbf{w})}_{\text{current value}}}_{\text{TD error}}\Big)\nabla_\mathbf{w}\hat{q}(S,A,\mathbf{w})$$

Q-Learning Update [1]

[1] V. Mnih et al., “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

Value-Based and Policy-Based Methods

Value-Based Methods:

Interaction → Estimate the Action-Value Function → Optimal Policy

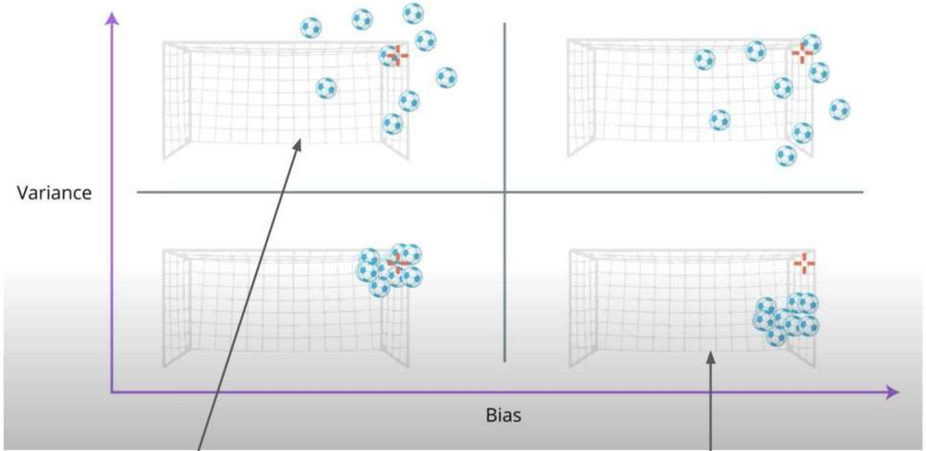
Policy-Based Methods:

Interaction → ~~Estimate Value Functions~~ → Optimal Policy

Why Policy-Based Methods?

1. Simplicity *(A direct mapping from the environment's state to the agent's action)*
2. Stochastic policies
3. Continuous action space

Actor-Critic Methods



Policy-based agents:
lower bias but higher variance

Value-based agents:
higher bias, but lower variance

Actor-Critic: A trade-off between value-based and policy-based RL agents

Actor:

- Policy-based
- Learn to make a good decision (evaluated based on metrics of the critic)

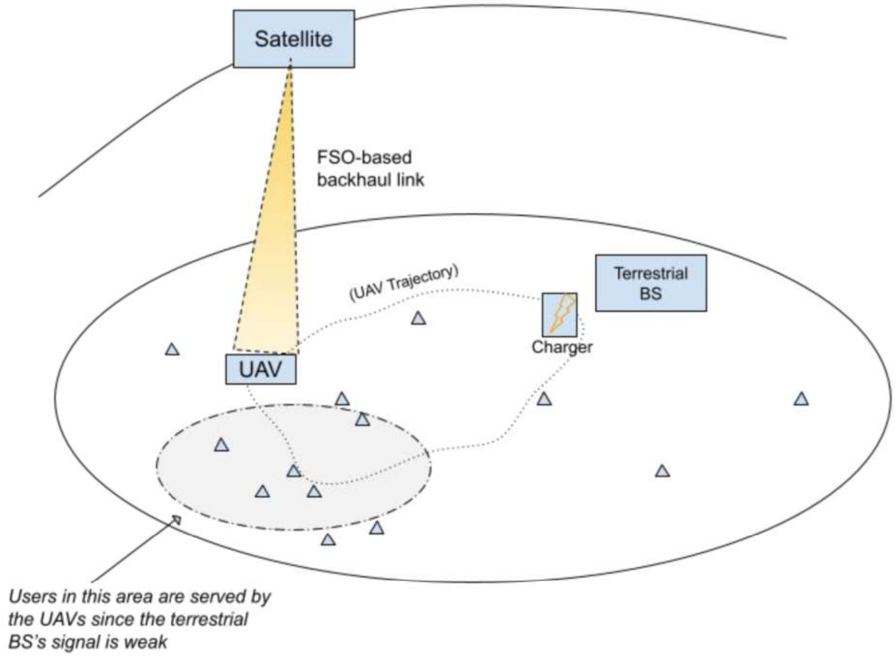
Critic:

- Value-based
- Learn to estimate (predict) the state-value function using the TD estimate

Part II: RL for UAV-BS Deployment

A Scenario for UAV-BS Deployment

One UAV-BS is deployed to complement the terrestrial BS (macro BS).



Methodology: A2C Algorithm (1/2)

A2C = Advantage Actor-Critic
(the synchronous version of the A3C [1]
--Asynchronous Advantage Actor-Critic)

Actor: outputs logits for a categorical probability distribution over all possible actions.

Critic: estimates the state-value function of the environment's state.

[1] V. Mnih et al., "Asynchronous methods for deep reinforcement learning," in Proceedings of the 33rd International Conference on Machine Learning (ICML), vol. 48, 2016, p. 1928–1937.

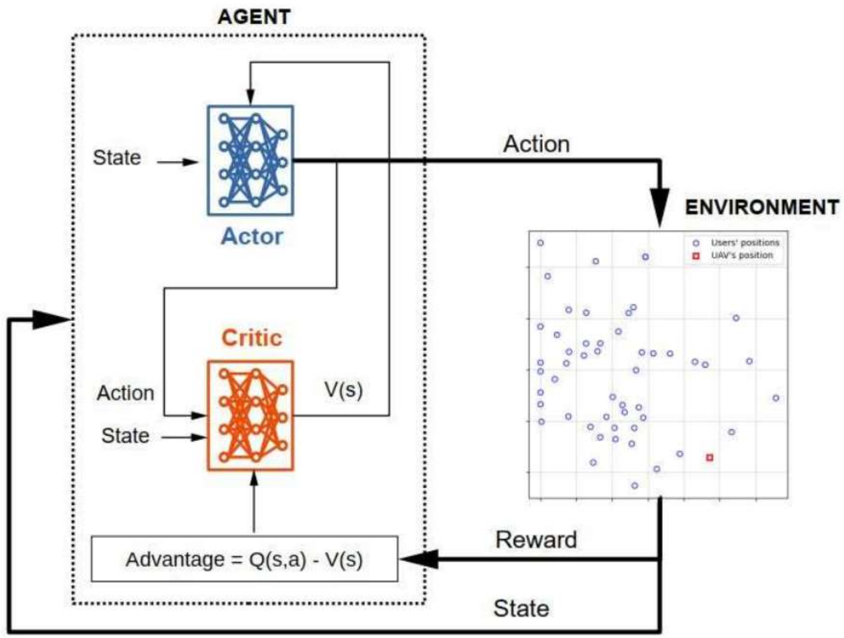


Figure 2: A2C Method

Methodology: A2C Algorithm (2/2)

- **State**: The users' and UAV-BS's coordinates
- **Actions**: move to the (1)-north, (2)-west, (3)-south, (4)-east, or (5)-remain stationary (no movement)
- **Reward**:
  +1 if d(t+1) > d(t),
  -1 if d(t+1) < d(t),
  -0.1 if d(t+1) = d(t)

(d(t): the average data rate of all users at time t)

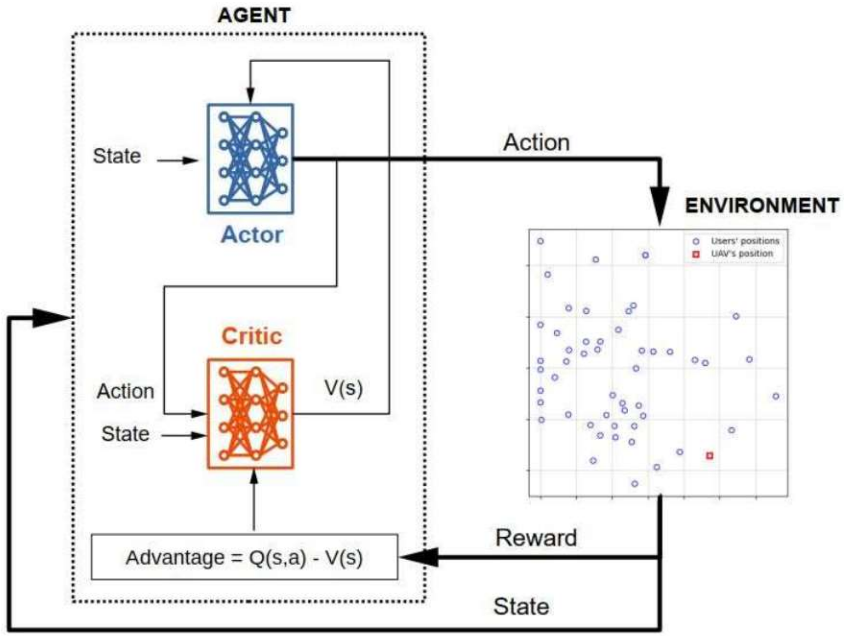
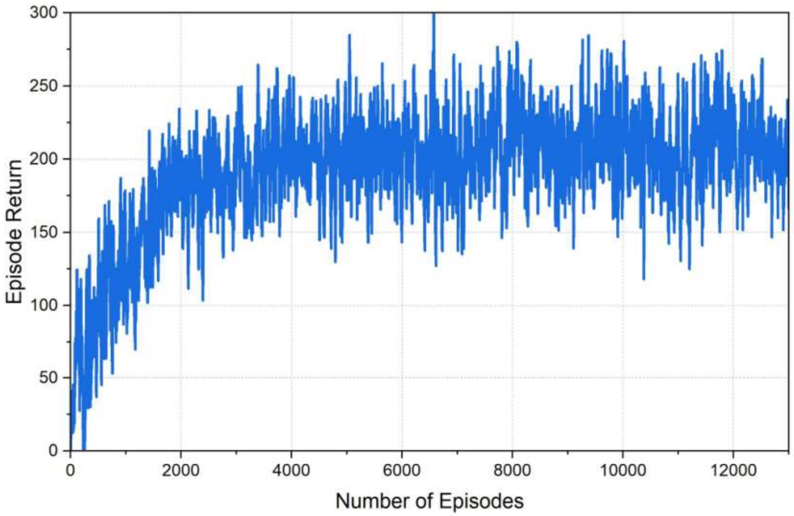


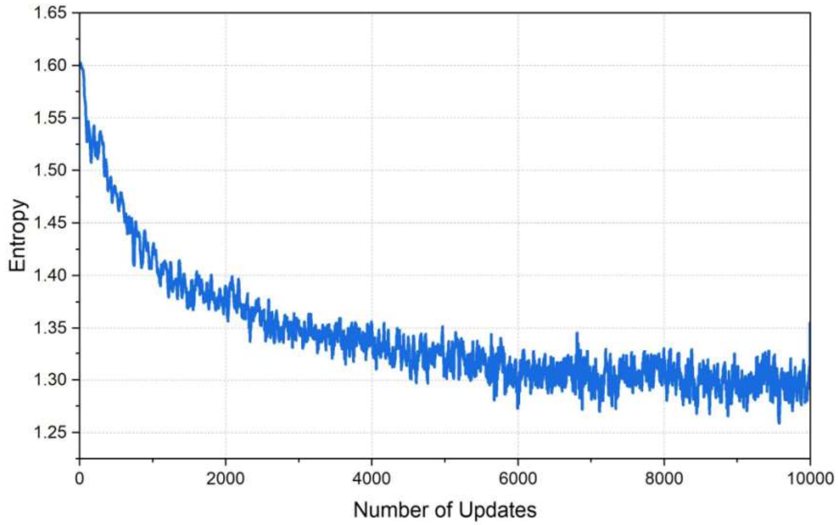
Figure 2: A2C Method

Initial Results (1/2)

The agent gradually forms better movement policy for the UAV-BS with higher rewards



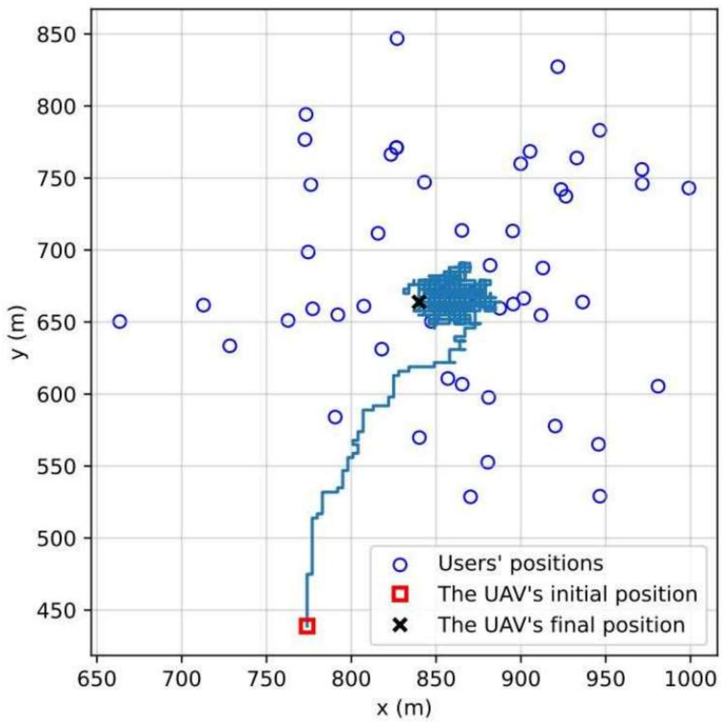
The action selected by the agent gradually becomes less random (i.e., more intentional)



Initial Results (2/2)

To demonstrate the generalization ability of the agent after training, a trained A2C agent is tested to control the UAV-BS's movements in an episode.

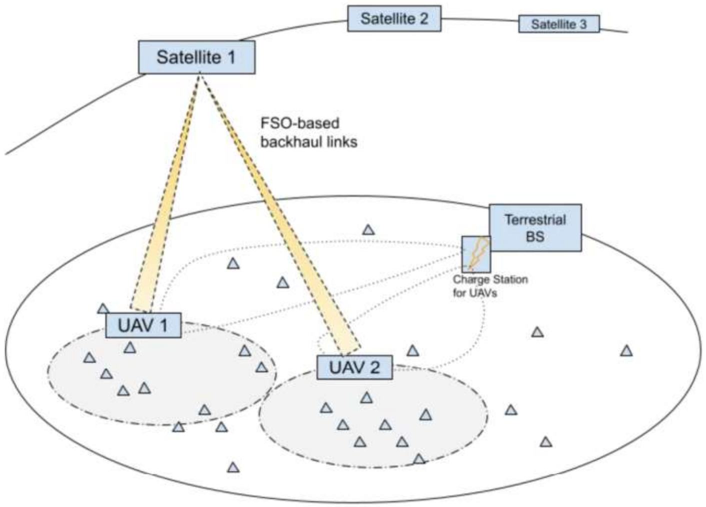
- The test environment (i.e., the spatial distribution of users and the UAV's initial location) was set up randomly and not previously known by the agent.
- The agent was not trained during the test.

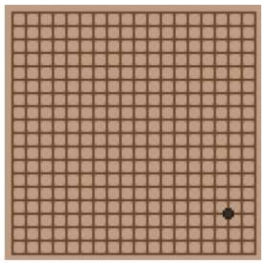


Behavior of an A2C agent after 4 hours of training

The Road Ahead

- **Satellite-Air-Ground Integrated Networks (SAGIN)**: multiple UAV-BSs are deployed to complement the terrestrial BS.
- Take into account constraints of the **FSO-based backhaul links** with LEO satellites.
- Multiple UAV-BSs are expected to cooperate to efficiently serve the ground users → **Multi-agent RL**





Multi-agent RL for playing Go

References

1. R. S. Sutton and A. G. Barto, Reinforcement Learning, 2nd ed. The MIT Press, 2018.
2. V. Mnih et al., “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.
3. N. Heess et al., “Emergence of Locomotion Behaviours in Rich Environments.” arXiv, 2017. doi: 10.48550/ARXIV.1707.02286.
4. J. Liu, Y. Shi, Z. Md. Fadlullah, and N. Kato, “Space-Air-Ground Integrated Network: A Survey,” *IEEE Commun. Surv. Tutorials*, vol. 20, no. 4, pp. 2714–2741, 2018.
5. A. Feriani and E. Hossain, “Single and Multi-Agent Deep Reinforcement Learning for AI-Enabled Wireless Networks: A Tutorial,” *IEEE Commun. Surv. Tutorials*, vol. 23, no. 2, pp. 1226–1252, 2021.

Thank you for your attention!