Harnessing AI for Sustainable Agriculture and Climate Adaptation: A Resilience Proposal

Fay Majid Elhassan
Aims-Senegal
elhassan@aimsammi.org

Abstract

In the pursuit of sustainable agriculture, greenhouses provide a controlled setting ripe for automation innovations. This study delves into the potential of reinforcement learning (RL) to autonomously fine-tune greenhouse conditions. We meticulously evaluate Proximal Policy Optimization (PPO), Asynchronous Advantage Actor-Critic (A2C), and Soft Actor-Critic (SAC) in a simulated greenhouse context, highlighting the nuances of each algorithm's performance. The results underscore the adaptability and proficiency of RL in orchestrating complex parameters like water regulation and thermal management. Beyond the immediate findings, this research paves the way for harnessing RL in broader agricultural contexts, emphasizing its role in future sustainable farming solutions

1 Introduction

With the rising global food demands and the challenges posed by climate change, greenhouses have become a cornerstone in sustainable agriculture [1]. This study explores the potential of RL as a pivotal tool for greenhouse management optimization.

Merging artificial intelligence (AI) capabilities with real-time data, we aspire to lead breakthroughs in sustainable agriculture and energy conservation. This research, with its global implications, emphasizes community well-being and environmental protection as we collectively forge a path to a brighter, more resilient future.

1.1 Contributions

Our key contributions encompass an RL-driven approach to greenhouse control, the creation of a realistic simulated greenhouse environment, a comprehensive analysis of three premier RL algorithms within the greenhouse control context, and insights into their real-world performance and implications.

The paper's structure is as follows: **Section 2** delves into RL's role in greenhouse control, **Section 3** offers insights into our greenhouse simulation, **Section 4** discusses the experimental setup, **Section 5** presents our results and comparative analysis, **Section 6** deliberates on our findings and potential future directions, and **Section 8** concludes with broader implications and visions.

Join us in this exploration of RL's intricacies within greenhouse management, as we chart a course towards a sustainable future.

2 Background: Reinforcement Learning in Advanced Agricultural Systems

RL has emerged as a potent tool in various domains, including advanced agricultural systems. At its core, RL revolves around agents learning to make decisions by interacting with their environment.[2] This section offers a deep dive into the foundational concepts of RL, the nuances of deep RL methods, and their relevance to agricultural systems.

2.1 Reinforcement Learning Problem Statement

RL operates on the principles of the Markov Decision Process (MDP)[3], a framework for decision-making in dynamic environments. In MDPs, an agent observes a state s, selects an action a, obtains a reward r, and transitions to a new state s'. The agent's behavior is guided by a policy π, which associates states with probable actions. The discount factor γ, usually in the range [0,1], determines the significance of future rewards. The ultimate aim of RL is to determine the best policy that maximizes the cumulative reward over time.

2.2 Exploration vs. Exploitation

In RL, an agent must find a balance between exploration (trying new actions) and exploitation (sticking with known actions). The parameter ϵ governs this balance. An ϵ-greedy strategy will choose a random action with probability ϵ and the action currently believed to be the best with probability $1 - \epsilon$.

2.3 Deep Reinforcement Learning

Deep reinforcement learning (DRL) merges the power of deep learning with traditional RL. It uses neural networks as function approximators to handle large state and action spaces, making it apt for complex tasks like greenhouse management.[4]

- **Policy Gradients**: Policy gradient methods optimize the policy directly by ascending the gradient of the expected return. These methods have the advantage of being effective in high-dimensional, continuous action spaces.
- **On-policy vs. Off-policy Algorithms**:
  - *On-policy algorithms* learn about the current policy being used and use the same policy to select actions.
  - *Off-policy algorithms* learn about a policy different from the one used to select actions.

2.4 Reinforcement Learning in Greenhouses

RL is rooted in the interplay of agents, environments, states, actions, and rewards:[2]

- **Agents**: Decision-makers that learn from their interactions.
- **Environments**: The context of interaction, represented by the greenhouse in our study.
- **States**: Current conditions, e.g., temperature or humidity in a greenhouse.
- **Actions**: Decisions by agents, such as adjusting ventilation in greenhouses.
- **Rewards**: Feedback on the agent's decisions, with positive values for optimal actions and negative for suboptimal ones.

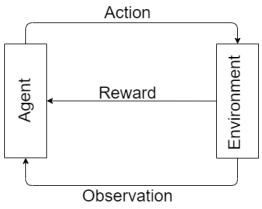

Figure 1: Reinforcement Learning Parts

2.5 Relevance to Greenhouse Control

Greenhouse control, like many RL problems, can be framed within the Markov Decision Processes (MDP) paradigm [3]. The Markov property, intrinsic to MDPs, signifies that future states depend solely on the current state and action. This principle is pivotal in greenhouses where immediate states, such as soil moisture, are influenced by present actions and conditions.

Considering the multifaceted dynamics of greenhouses, from fluctuating weather to unique plant needs, RL's adaptive learning offers a promising alternative to traditional rule-based systems, which may fall short in managing these intricacies. Our study focuses on three RL algorithms, namely PPO, A2C, and SAC, evaluating their aptness and efficacy in greenhouse management.

3 Methods

3.1 Simulated Greenhouse Environment

Our simulation replicates the intricate dynamics of real-world greenhouses. The state space derives from greenhouse observations, capturing essential parameters like temperature, humidity, light intensity, and CO2 concentration. Each state is represented as a vector, providing the RL agent with a comprehensive view of the greenhouse's conditions. Our study hinges on merging agriculture with RL, using the Autonomous Greenhouse Challenge dataset[5]. This dataset enables us to model how tomato crop knowledge can be encoded into an AI algorithm for automated, human-like decisions.

3.2 Action Space

The action space dictates the agent's greenhouse control options. We studied:

- **Discrete Action Space**: The agent picks from set actions, e.g., 100 mL or 200 mL.
- **Continuous Action Space**: The agent selects any value within bounds, e.g., 0 to 1000 mL.

3.2.1 Reward Mechanism

Driving the agent towards optimal greenhouse operations is our reward structure. Immediate feedback is provided at each timestep based on the agent's actions. When the agent's choice mirrors the estimated optimal action for the prevalent state, it's rewarded positively. The reward's magnitude is inversely tied to the deviation between the agent's choice and the estimated best action. Such a design ensures the agent's motivations align with the greenhouse's overall health and productivity.

Given:

- a - Action taken by the agent.
- a^* - Estimated optimal action for the current state.

The reward $R(a)$ in Assimilation Setpoint Control environoment and Heat Control is defined as:

$$R(a) = \begin{cases} 100 & \text{if } a = a^* \\ -1 \times |a - a^*| & \text{otherwise} \end{cases}$$

While in Water Level Control is defined as :

$$R(a) = \begin{cases} 1000 & \text{if } a = a^* \\ -1 \times |a - a^*| & \text{otherwise} \end{cases}$$

This means that the agent receives a reward of 100 or 1000 depending on space its exposed to when its action matches the estimated optimal action. Otherwise, it receives a penalty proportional to the deviation from the optimal action.

3.2.2 Challenges Encountered

1. **Continuous Action Spaces**: The vast possibilities in continuous spaces necessitate sophisticated methods for approximating optimal action-values.
2. **Greenhouse Complexity**: The non-linear relationships among greenhouse parameters make prediction difficult, requiring the agent to grasp intricate dynamics.
3. **MacBook M1 Limitations**: Continuous training of advanced RL algorithms overextended periods led to overheating and reduced battery life on the MacBook M1.

4. **Memory Issues**: The extensive environment scale and numerous training steps resulted in memory and data storage challenges.

These challenges, while substantial, provided invaluable insights and learning experiences, shaping the trajectory of our research and the methodologies employed.

3.3 Algorithms

We employed three primary reinforcement learning algorithms tailored for the greenhouse environment:

- **PPO** : Ensures stable learning by limiting policy updates, paired with a discretized action space.[6]
- **A2C** : Uses the actor-critic structure with parallel instances for faster learning, combined with a discretized action space.[7]
- **SAC** : Designed for continuous actions, it promotes exploration through entropy.[7]

4 Experiments

4.1 Setup

Our experiments revolved around a simulated greenhouse environment. The primary task delegated to the RL agent was to optimize the control strategies for three main parameters: water level, heat, and assimilation setpoint.

4.1.1 Pre-processing Steps

- **Normalization**: All observation data underwent normalization to ensure that all parameters, irrespective of their original scales, converged to a similar range. This pre-processing step facilitated the RL agent in discerning patterns and relationships more effectively.
- **Action Space Initialization**: Depending on the specific requirements of each experiment, the action spaces were initialized as either continuous or discrete.

4.1.2 Training Methodology

In our simulated greenhouse, RL agents underwent extensive training, often exceeding 5000 timesteps and lasting over five days to stabilize.

Training utilized a dataset of 47,000 observations across five sets, gathered over 12 weeks [5]. The agent began with the first set's data, predicting actions based on observations. Rewards or penalties were assigned based on prediction accuracy, with greater deviations incurring larger penalties. This training looped through all datasets.

Training termination was decided when epsilon and reward values stabilized, signifying reduced agent exploration.

4.2 Implementation Details

The Stable Baselines3 library [7]served as the foundation for our experimental setup. see Appendix A for further details.

5 Results

5.1 Water Level Control

Water, being vital for plant growth, requires precision in its management within greenhouse settings. Our algorithms displayed commendable performance in maintaining optimal water levels, showcasing consistent rewards and stability.

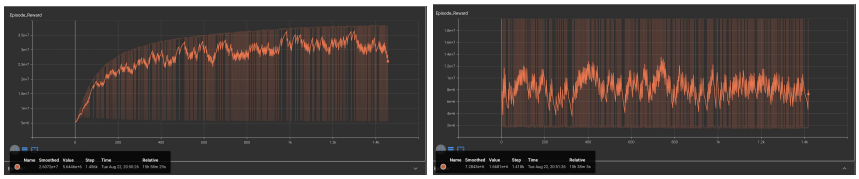

Figure 2: Performance of PPO (left) and A2C (right) in water space.

PPO exhibits a stabilizing trend, achieving consistent rewards after initial timesteps. In contrast, A2C demonstrates noisier learning with significant fluctuations, although it reaches a reasonable average reward.2 for a visual representation.

5.2 Heat Control

Temperature regulation plays a pivotal role in plant health. The algorithms' adaptability and precision were tested in modulating the greenhouse's temperature. As noted, in spaces like heat, some algorithms faced challenges during the learning process, leading to fluctuations in their performance. The results, depicted in Figure 3, shed light on their proficiency in this crucial domain. Both

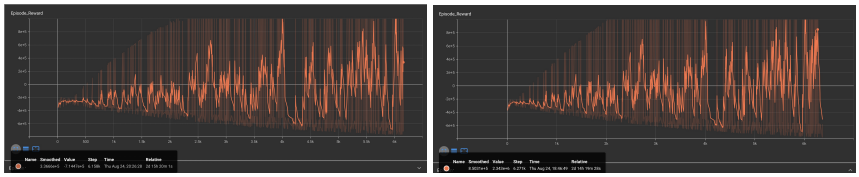

Figure 3: Performance of PPO (left) and A2C (right) in heat space.

algorithms face challenges in the heat space. PPO presents an upward trend with fluctuations, indicating some struggles. A2C, on the other hand, displays more significant oscillations, suggesting challenges in policy stabilization.Figure above 3

5.3 Assimilation Setpoint Control (Discrete/Continuous)

Diving into the complexities of photosynthesis, the results for this metric were revealing. They highlighted each algorithm's capability to optimize the crucial aspect of plant growth. Despite initial fluctuations, a steady increase in rewards was observed, dependent on the algorithm used. The results are illustrated in Figure 4.

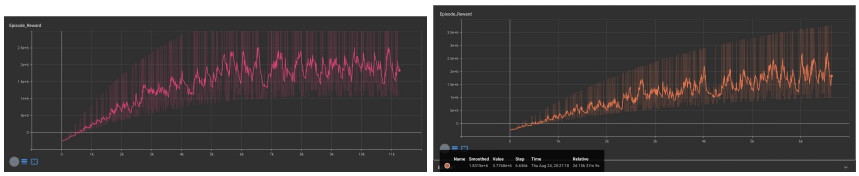

Figure 4: Performance of PPO (left) and A2C (right) in assimilation space (discrete).

PPO's performance in this space is marked by a steady increase in reward with minor fluctuations. A2C, while exploring the space effectively, showcases a more volatile learning curve with pronounced spikes and drops.

SAC showcases an upward learning trend with occasional fluctuations. PPO, in the same space, demonstrates a more consistent learning curve, achieving higher rewards over time.see Figure 5

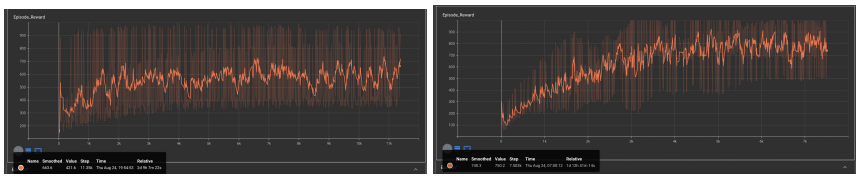

Figure 5: Performance of SAC (left) and PPO (right) in assimilation space (continuous).

6 Discussion and Conclusion

Building upon foundational work that introduced DQN to greenhouse control[8], our study delved into the intricacies of PPO, A2C, and SAC within varied greenhouse scenarios.

PPO shone brightly in both continuous and discrete spaces, demonstrating swift learning and stability due to its 'actor-critic' architecture, tailored for dynamic greenhouse environments.

A2C, nimble in discrete space, faced variability challenges. Notably, in the 'heat' environment, A2C underwent significant fluctuations, underscoring the complexities of this domain. Yet, its equilibrium between exploration and exploitation remains promising.

SAC, in the continuous action domain, exhibited steady learning, but its computational intensity can pose challenges. Concurrently, PPO showcased versatility in this space.

In examining specific scenarios, the 'water' domain was more accommodating, with algorithms reliably garnering rewards. Conversely, the 'assimilation' domain had initial hiccups but later stabilized. The 'heat' space was particularly tumultuous for both PPO and A2C, indicating its inherent challenges.

Contrasted with the earlier DQN approach[8], our results emphasize the superior capabilities of the studied algorithms, especially PPO. With better computations resources this research paves a promising trajectory for future RL endeavors in sustainable agricultural automation, hinting at a bright future for precision agriculture advancements.

References

[1] Wageningen University & Research (WUR). "Greenhouse Horticulture, Wageningen University & Research (WUR), 6708PB Wageningen, The Netherlands. Author to whom correspondence should be addressed." In: *Sensors* 20.22 (2020). Received: 21 October 2020 / Revised: 3 November 2020 / Accepted: 6 November 2020 / Published: 11 November 2020, p. 6430. DOI: 10.3390/s20226430.

[2] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[3] Oguzhan Alagoz et al. "Markov decision processes: a tool for sequential decision making under uncertainty". In: *Medical Decision Making* 30.4 (2010), pp. 474–483.

[4] Peter Henderson et al. "Deep reinforcement learning that matters". In: *arXiv preprint arXiv:1709.06560* (2017).

[5] Wageningen University & Research. *Autonomous Greenhouses - 2nd Edition*. Accessed: yyyy-mm-dd. URL: https://www.wur.nl/en/project/autonomous-greenhouses-2nd-edition.htm.

[6] J. Schulman et al. *Proximal Policy Optimization Algorithms*. URL: joschu,%20filip,%20prafulla,%20alec,%20oleg%20@openai.com.

[7] *Stable Baselines3 Algorithms*. Accessed: July-2023/August-2023. URL: https://stable-baselines3.readthedocs.io/en/master/guide/algos.html.

[8] K. Kaushal. *Autonomous Greenhouse Control*. 2021. URL: https://github.com/Kaushal1011/AutonomousGreenhouseControl/tree/main.

Appendix

A Implementation Details and Pseudocode

For the distinct scenarios of ASSIM, HEAT, and WATER, both the PPO and (A2C) algorithms were employed on discrete action spaces while SAC and PPO was implemented on continuous. The hyperparameters were consistent across all these experimental scenarios.

A.1 Environment: AGCRLEnv (Continuous Action Space)

The custom AGCRLEnv environment simulates greenhouse conditions, enabling agents to decide on actions from a continuous spectrum.

Key Methods:

- **step**: Processes the agent's action, yielding the subsequent observation, reward, and the episode's completion status.
- **map_continuous_to_interval**: Transforms a continuous action into its corresponding interval within the action domain.
- **rewardfunc**: Allocates feedback based on the chosen action's proximity to the target setpoints.
- **estimate_closest_as**: Approximates the nearest interval for a given value in the action space.
- **reset**: Reinitializes the environment for the upcoming episode. If required, it transitions to the subsequent team's dataset.

Pseudocode:

Algorithm 1 AGCRLEnv Continuous Action Space Pseudocode

Initialization:
    a. Define continuous action and observation space.
    b. Load observations and actions data.
    c. Set starting indexes for team and observation.
function STEP(action)
    a. Map the continuous action to a specific interval using MAP_CONTINUOUS_TO_INTERVAL(action).
    b. Calculate reward using REWARDFUNC(action).
    c. Increment observation index.
    d. Update current and next observations.
    e. Check if episode is done (when observations for the team are exhausted).
    f. Return observation, reward, and done flag.
end function
function MAP_CONTINUOUS_TO_INTERVAL(action)
    Convert the continuous action to its corresponding interval in the action space.
end function
function REWARDFUNC(action)
    a. Compare the predicted action to the real action.
    b. Return a high reward for close matches and a penalty for deviations.
end function
function ESTIMATE_CLOSEST_AS(value)
    Estimate the closest interval in the action space to the given value.
end function
function RESET
    a. Move to the next team's data.
    b. Reset observation index.
    c. Update observation space to the current team's starting observation.
end function

A.2 Environment: AGCRLEnv (Discrete Action Space)

This custom environment emulates the conditions of a greenhouse, allowing agents to select actions within a discrete set of choices.

Key Methods:

- **step**: Processes the agent's action, returns the new observation, reward, and completion status.
- **rewardfunc**: Provides feedback based on the proximity of the chosen action to the desired setpoints.
- **estimate_closest_as**: Estimates the closest discrete action for a given value in the action space.
- **reset**: Resets the environment to a new episode, moving to the next team's data if necessary.

Pseudocode:

Algorithm 2 AGCRLEnv Discrete Action Space Pseudocode

Initialization:
    a. Define action and observation space.
    b. Load observations and actions data.
    c. Set starting indexes for team and observation.
function STEP(action)
    a. Calculate reward using REWARDFUNC(action).
    b. Increment observation index.
    c. Update current and next observations.
    d. Check if episode is done (when observations for the team are exhausted).
    e. Return observation, reward, and done flag.
end function
function RESET
    a. Move to the next team's data.
    b. Reset observation index.
    c. Update observation space to the current team's starting observation.
end function
function REWARDFUNC(action)
    a. Estimate the closest action space value to the actual value.
    b. If the predicted action matches, return a high reward.
    c. Otherwise, return a penalty proportional to the prediction error.
end function
function ESTIMATE_CLOSEST_AS(value)
    Estimate the closest action in the action space to the given value.
end function
function RESETINIT
    a. Reset to the first team's data.
    b. Reset observation index.
    c. Update observation space to the first team's starting observation.
end function

A.3 Soft Actor-Critic (SAC)

SAC is an off-policy actor-critic DRL algorithm based on the maximum entropy RL framework. It emphasizes not only maximizing expected returns but also the entropy of the policy, promoting exploration.The Soft Actor-Critic (SAC) algorithm was employed to optimize greenhouse management. SAC's inherent ability to work with continuous action spaces made it a fitting choice for tasks like assimilation setpoint control. The code provided leverages the Stable Baselines 3 library, emphasizing the continuous action space, exploratory settings, and the application of TensorBoard for tracking.

Hyperparameters:

- Learning Rate: 0.003
- Entropy Coefficient: 0
- Gradient Steps: -1 (Auto-adjust)

Pseudocode:

Algorithm 3 SAC Pseudocode

```
Define the action space for assimilation setpoint control.
Initialize the custom greenhouse environment 'AGCRLEnv'.
Wrap the environment for vectorized operations and normalization.
Define exploratory settings with epsilon decay.
Initialize SAC agent with given hyperparameters.
Set up TensorBoard for monitoring.
while True (Continuous Training Loop) do
    a. Reset environment at the beginning of each episode.
    while For each step within an episode do
        i. Decide on action (either predict using SAC or explore based on epsilon).
        ii. Execute action and observe reward and next state.
        iii. Accumulate reward for the episode.
    end while
    c. Update the SAC model periodically.
    d. Log necessary details to TensorBoard.
    e. Save model checkpoints based on reward thresholds.
end while
```

A.4 Advantage Actor-Critic (A2C)

A2C is an on-policy algorithm leveraging an advantage function to minimize the variance of policy gradient updates.The A2C algorithm, a synchronous and deterministic variant of the A3C, was employed for its proficiency in handling discrete action spaces. Using the Stable Baselines 3 library, our A2C approach managed the greenhouse environment. With an episodic training loop, the agent's policy was updated using both value and policy loss, and TensorBoard was leveraged for in-depth monitoring.

Hyperparameters:

- Learning Rate: 0.003
- Epsilon (for exploration): Decayed from 1 to 0.15 using a decay factor of 0.99975

Pseudocode:

Algorithm 4 A2C Pseudocode

```
Define environment settings.
Initialize A2C agent with desired hyperparameters.
Set up TensorBoard for monitoring.
Define exploration settings with epsilon decay.
while True (Continuous Training Loop) do
    a. Reset environment at the start of each episode.
    while Within each episode do
        i. Decide on action (either predict using A2C or explore based on epsilon).
        ii. Execute action and observe reward and next state.
        iii. Accumulate episode reward.
    end while
    c. Update the A2C model at the end of each episode.
    d. Log relevant metrics to TensorBoard.
    e. Save model checkpoints based on reward criteria.
end while
6. Plot results for episodic rewards and accumulated regret.
```

A.5 Proximal Policy Optimization (PPO)

PPO, an on-policy algorithm, introduces minor policy updates to deter drastic alterations that could be detrimental to the learning trajectory.PPO is known for its robustness and efficacy across various environments. In the context of greenhouse management, PPO was utilized for its balance between exploration and exploitation, ensured by the clipping mechanism. The Stable Baselines 3 library was employed for a streamlined PPO implementation. A primary advantage of PPO in our setup was its capability to handle both discrete and continuous action spaces. Moreover, PPO's policy iteration process ensured smooth learning curves without drastic policy updates, which is paramount in environments like greenhouses where drastic actions can lead to detrimental outcomes.

Hyperparameters:

- Learning Rate: 0.003
- Batch Size: 1024
- Clip Range: 0.15
- Epsilon (for exploration): Decayed from 1 to 0.15 using a decay factor of 0.99975

Pseudocode:

Algorithm 5 PPO Pseudocode

Define environment and exploration settings.
Initialize PPO agent with specific hyperparameters.
Set up TensorBoard for monitoring.
while True (Continuous Training Loop) **do**
    a. Reset environment at the start of each episode.
    while Within each episode **do**
        i. Decide on action (either predict using PPO or explore based on epsilon).
        ii. Execute action and observe reward and next state.
        iii. Accumulate episode reward.
    end while
    c. Update the PPO model at the end of each episode.
    d. Save model checkpoints based on reward criteria.
    e. Log relevant metrics to TensorBoard.
end while
5. Plot results for episodic rewards and accumulated regret.