Deep Reinforcement Learning for Target Acquisition in a Hostile Environment

Aaron Brown, Alex Jordan, Grant Stagg

Theory of Predictive Modeling

CS 580/PHSCS 513R

Background

Autonomous agents have the potential to assist humans in a wide range of applications. Target acquisition is one potential application in which autonomy could provide high value. Effective autonomous target acquisition could be applied to situations such as search-and-rescue, fire fighting or military surveillance. In this work we present a method which uses reinforcement learning to train agents to locate targets while simultaneously avoiding potential dangers.

Problem Statement

Given a mobile agent in an unknown environment containing targets and enemies develop a steering policy that enables the agent to maximize the number of targets found while also avoiding enemies.

Each agent:
- Begins in the center of the environment with a random heading
- Has a knowledge of its state (x, y, heading) and sensor rays that scan for enemies and targets
- Optionally a grid map of where it has traveled
- Travels at a constant speed and controls its motion with choice of heading rate

Enemies:
- Scan the environment with a sinusoidal pattern, starting with some random phase offset
- "Catch" the agents if they come with in a designated radius and the simulation end

Targets:
- Are placed randomly within the environment at the start of each simulation
- Claimed by agents if they come within a designated radius

Markov Decision Process:
- We model the agent in the environment as a continuous Markov Decision Process (MDP) with states, actions and rewards at each timestep (S_t, A_t, R_t)
- The states S_t , are assumed to be the x, y and heading of the agent as well as a set of measurements from sensor rays for targets and for enemies. Also, optionally the environment is divided into a grid and each cell has a value of one if the agent has visited it and 0 otherwise
- The action A_t , is the agent's turn rate. We assume that state transitions are deterministic and follow the following equation: $\dot{S}t = \begin{bmatrix} v\cos\theta(t) \\ v\sin\theta(t) \\ u(t) \end{bmatrix}$
- We choose the reward signal to achieve the desired behavior:
  - A large negative reward is given if the agent leaves the boundary space
  - A large negative reward is given if the agent is caught by enemies
  - A large positive reward is given if the agent acquires a target
  - A small negative reward is given each timestep to incentivize rapid exploration
- The goal is then to find a policy $\pi(S_t)$, that selects an action A_t according to the current state such that the expected value of the sum of the discounted reward is maximized:

$$\pi^*(S_t) = argmax_{\pi(S_t)}(\mathbb{E}[\sum(\gamma^t R_t | A_t \sim \pi(S_t))])$$

- *Figure 1* shows an image of a typical environment.
- Targets are shown in green, enemies as pink diamonds, and our agent is black.
- Measurement rays are shown in red if an enemy is sensed and blue if a target is sensed.
- The blue cells represent the grid state, They are filled when the agent has visited that cell.

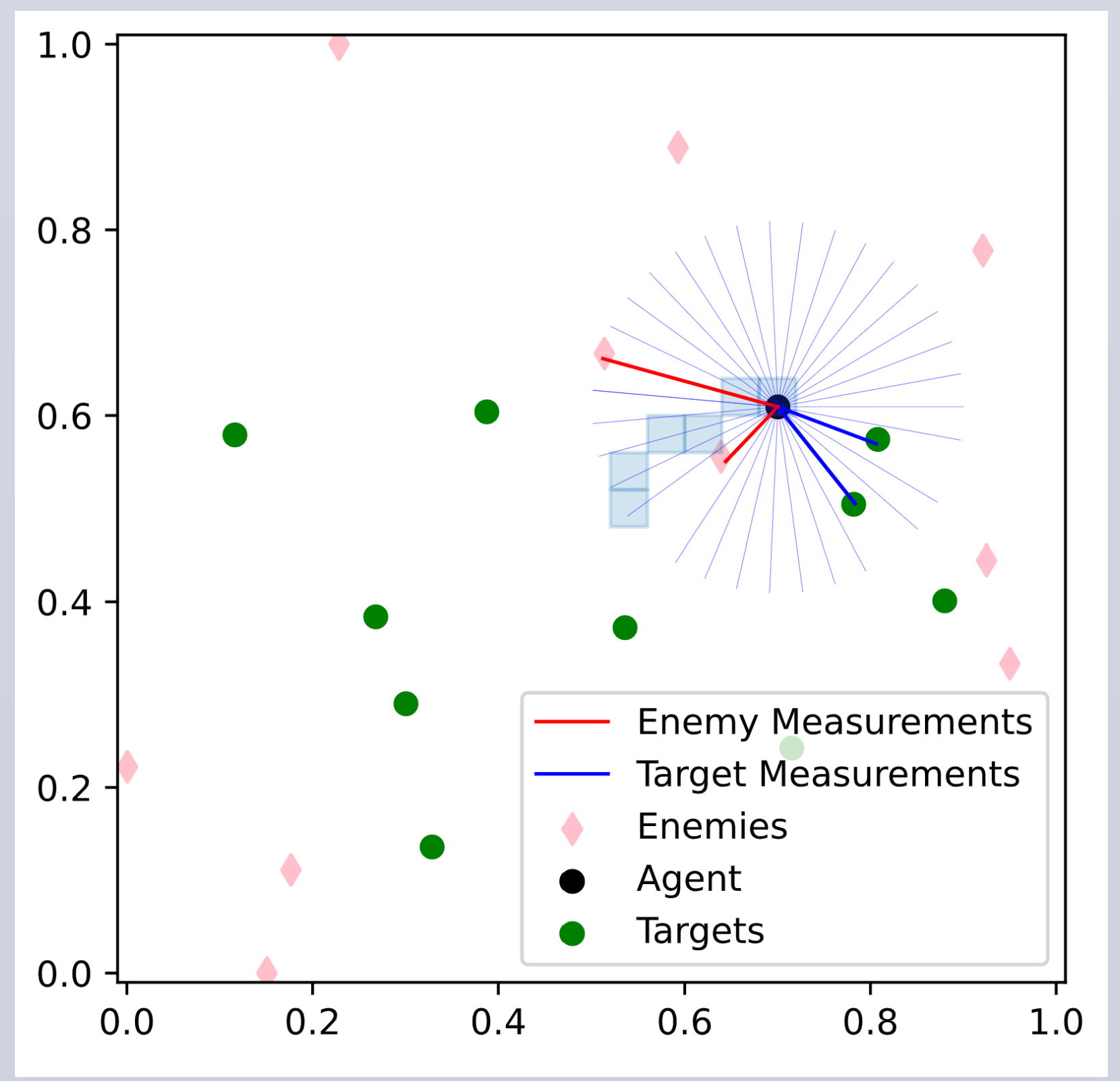

Figure 1: A sample environment.

Approach

- To approximate the optimal policy we used the Soft Actor Critic (SAC) algorithm.
- SAC uses separate neural networks to approximate the policy function $\pi(S_t)$, and the action, value function $q(S_t, A_t)$.
  - The policy network outputs a mean value and variance for the probability distribution of actions given a state. The action is then drawn from this distribution.
  - The action value network outputs the value of an action given the current state.
- SAC trains the policy network to maximize the expected sum of the discounted reward while simultaneously maximizing the policies entropy
  - This effectively creates the most random policy that is able to accomplish the task which incentives exploration during the learning process (which is critical to our problem because of the delayed reward of acquiring a target).
- We used two different network architectures for the different states (one without the grid cell states and one with the grid cell states)

No Grid Agent Networks:
- The input to the policy network is the x, y, heading of the agent as well as the sensor ray measurements for the targets and enemies.
- All states are flattened into a vector then passed through a neural network with two linear hidden layers with 64 nodes each and a rectified linear activation function.
- The action value network follows the same architecture as the policy network but concatenates the action A_t , with the state vector.

Grid Agent Networks:
- The grid is a two-dimensional array that describes where the agent(s) have been.
  - *Figure 1* shows that the agent has already visited six cells of the grid.
- The input for the policy network is the grid (2D array), in addition to the inputs of the no-grid agent networks mentioned above but are kept separate from the grid.
- The grid data first goes through a convolutional neural network on its own. This network contains two convolutions with max pooling in between, then is flattened to pass through three linear layers of descending size (see diagram), and a rectified linear activation function. (see *figure 2*)
- The output of this network then enters a new neural network with the other states.
  - This is similar to the no-grid agent network, except the layers are larger.
- The output of this network is the action of the agent.
- As with the no-grid agent network, the action value network follows the same architecture as the policy network, but outputs both the action and state vector.

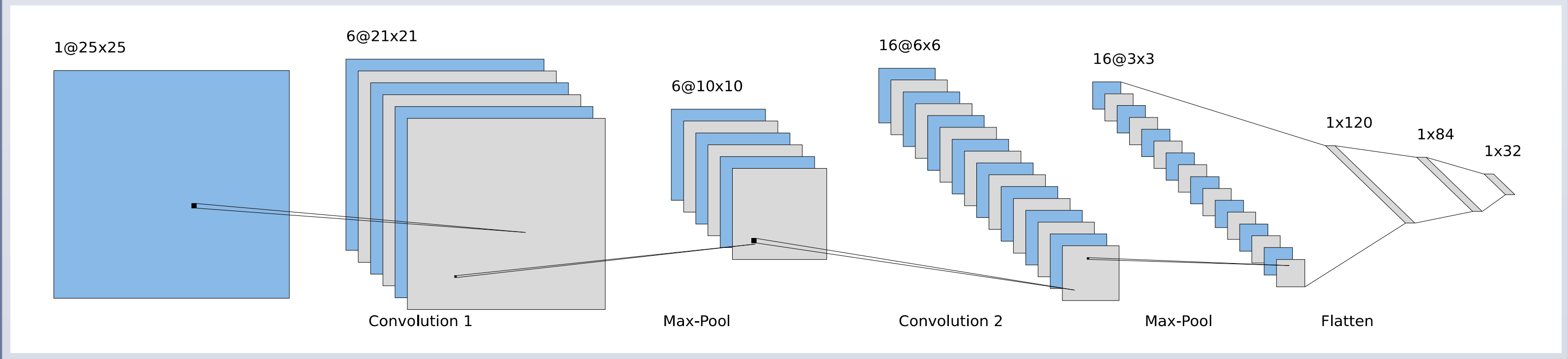

Figure 2: Diagram describing the framework of the Convolutional Neural Network used for the grid two-dimensional data

Results

- Trained agents were validated using Monte Carlo (MC) tests where the targets were placed in random locations for each MC run.
- None of the random locations used in training were repeated during validation.
- Trained agents were compared against two random baselines:
  - One agent with a random turn rate that could leave boundary space.
  - A random agent contained in the boundary by "bouncing" off of it.

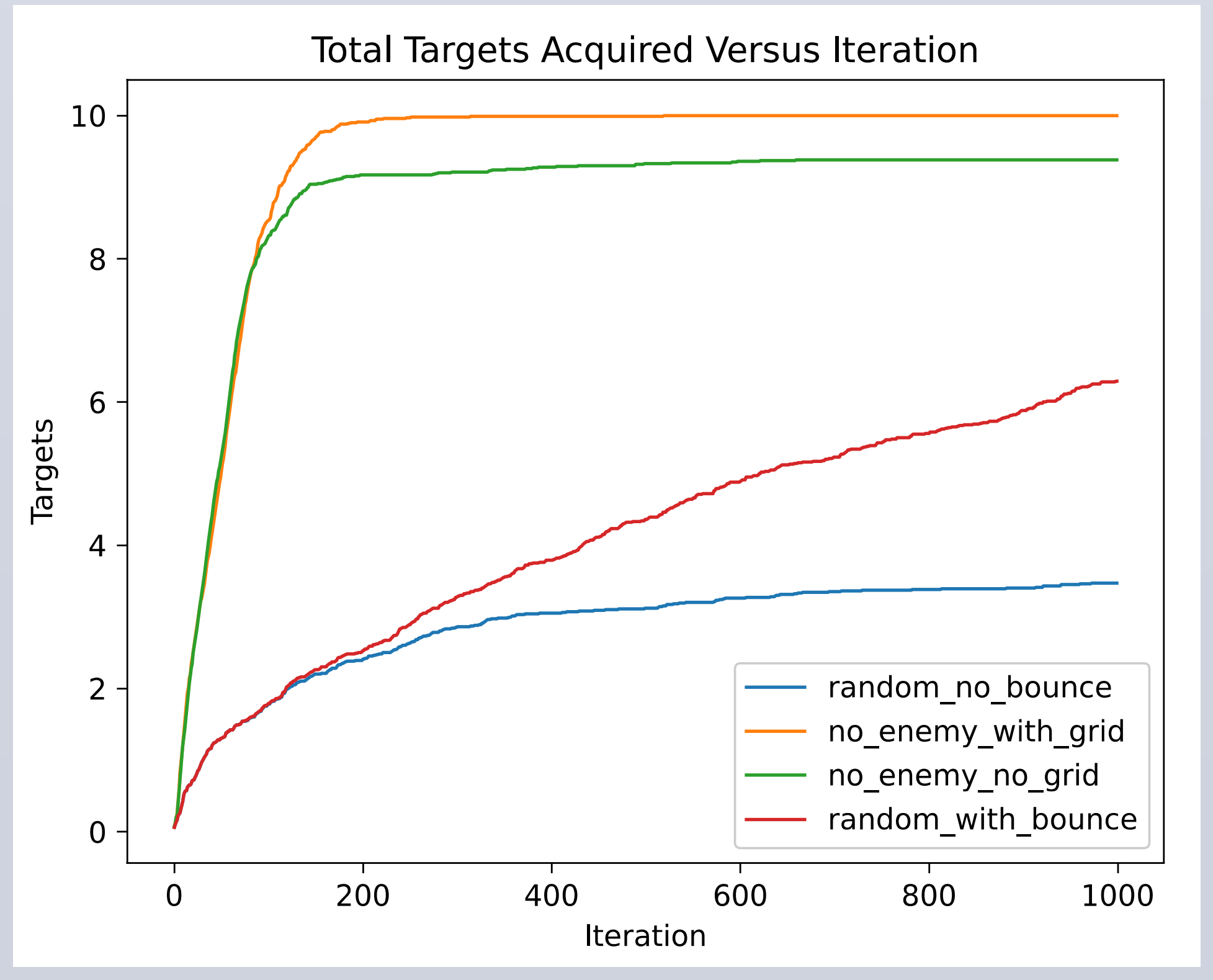

Figure 3: Average targets acquired versus iteration over 100 MC runs in environments with no enemies.

- Two tests were performed with 100 MC runs each:
  - Testing of agents trained without enemies
  - Testing of agents trained with enemies
- If an agent left the boundary the number of targets acquired remained the same for the rest of the time steps
- Both agents were trained and validated in enemy free environments
- The agent trained with the grid input was able to constantly find all 10 targets whereas the agent without the grid did not.

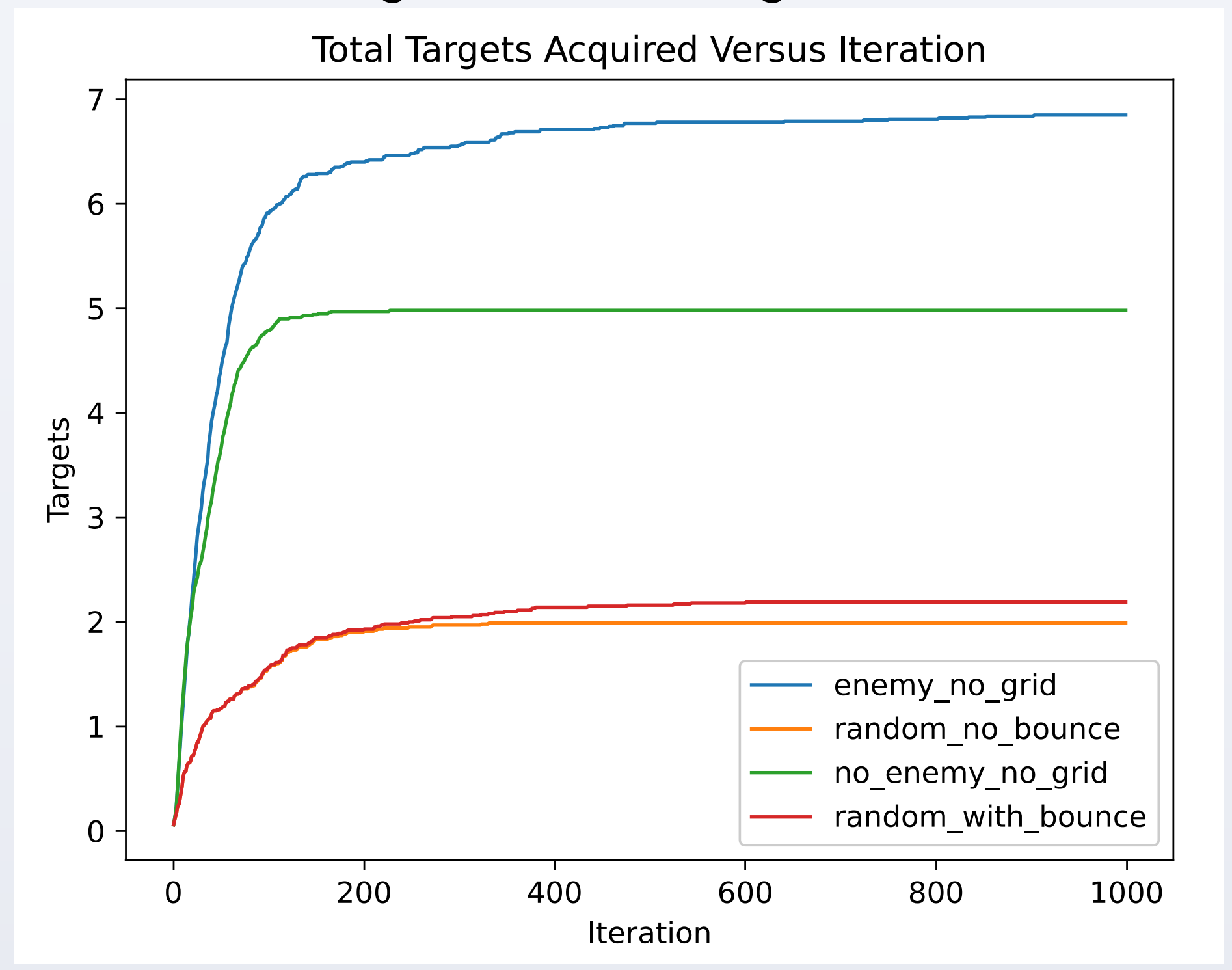

Figure 4: Average targets acquired versus iteration over 100 MC runs in environments with enemies.

- The agent trained to avoid enemies in an environment is not able to constantly find all 10 targets.
- The agent trained without enemies was unable to avoid enemies long enough to acquire more than 5 targets on average.
- We were unable to successfully train an agent to avoid enemies and seek targets with the grid method.

- Clearly, the agent trained to avoid enemies lasts longer than other agents. However, the enemies occasionally force the agent out of bounds resulting in shorter runs.

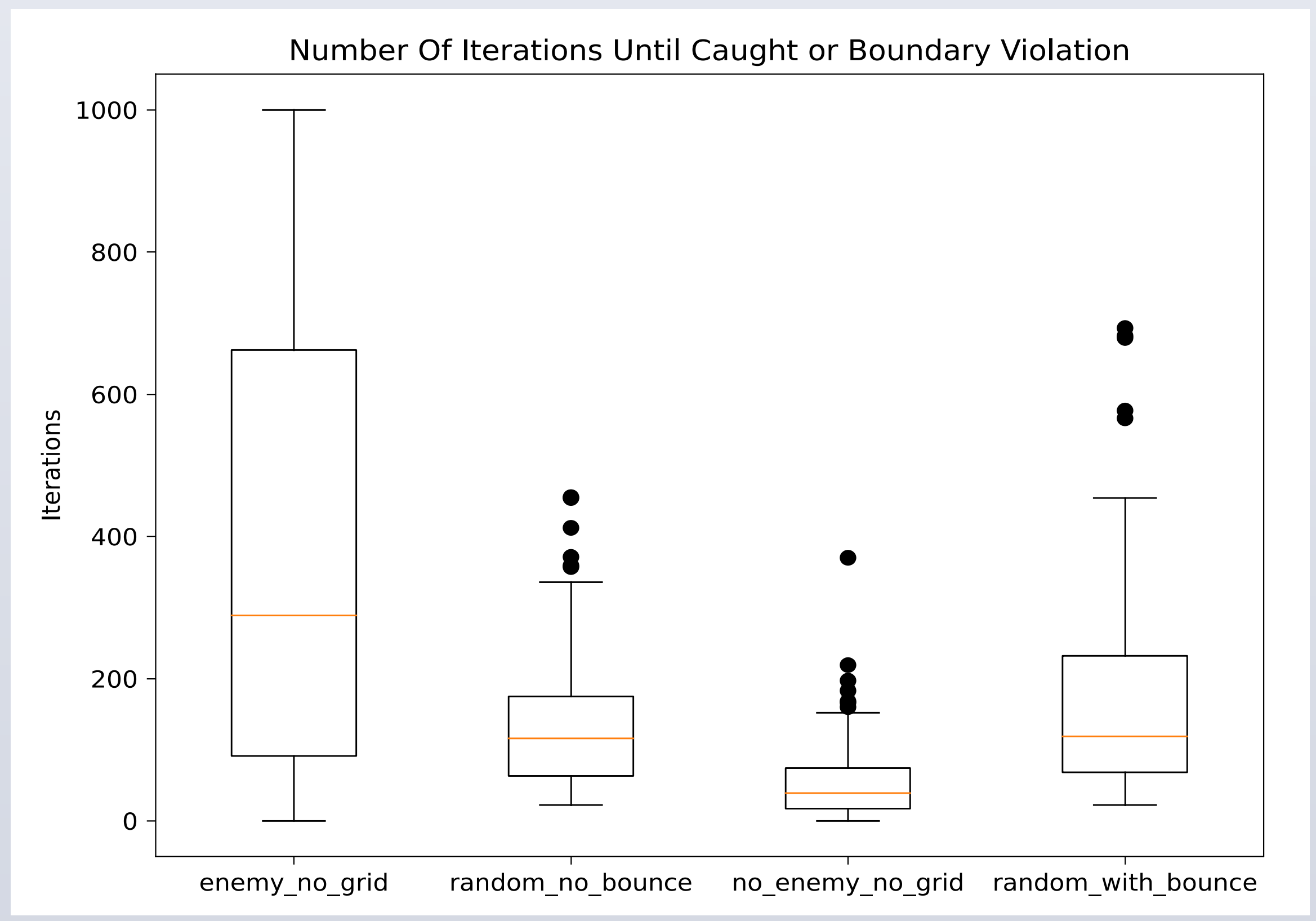

Figure 5: Iterations until agent is either caught by enemies or leaves the boundary space.

Conclusions

- We were able to successfully use the SAC algorithm to train an agent to stay within a boundary space, steer towards targets and avoid enemies.
- Giving the agent a sense of where it was been in the past helped the RL agent to better search for targets however it made training much more difficult.
- Most successful agents appeared to take a spiral path around the boundary space collecting targets and avoiding enemies along the way.
- Training was greatly affected by how the MDP was modeled (especially when choosing the rewards agents would get for specific actions).

Future Work

- Refine network architecture to make training easier and more consistent.
- Help agents learn to avoid situations where they will be forced out of bounds by enemies.
- Train "enemies" to learn optimal surveillance strategies to protect targets.
- Use multi-agent RL to train multiple agents/enemies to cooperate.