WITS UNIVERSITY

School of Computer Science and Applied Mathematics
UNIVERSITY OF THE WITWATERSRAND, JOHANNESBURG

Ecosystem Based Evolutionary Reinforcement Learning

COMS4030A

Michael Beukman Jesse Bristow Thabo Ranamane

June 22, 2021

Abstract

The ideas behind evolutionary algorithms and reinforcement learning are conceptually simple and intuitive. The fittest individuals pass on their genetic material to the next generation in the former and agents learn from their past experiences and actions in the latter. This report combines both of these methods into evolutionary reinforcement learning (ERL) and tests them in an open-ended ecosystem-based simulation.
We show that ERL is competitive with a simple reinforcement learning method and that interesting equilibria and structure can arise from simulating agents in an environment with simple rules.

1 Introduction

Artificial Life simulations have been popular for decades. They can be used to explore different learning algorithms (Ackley and Littman, 1992), investigate more complex individual based models of the real world (Huse *et al.*, 1999) and more. The main learning method used is usually some form of genetic algorithm, but methods such as reinforcement learning are also used (Sunehag *et al.*, 2019). There is also work that combines the two, where agents learn using reinforcement learning during their lifetimes and passes on the learned genes using evolutionary methods (Ackley and Littman, 1992; Khadka and Tumer, 2018).

We propose a simple simulation that can be used to investigate the relative performance of different learning methods. The simulation consists of prey, predators and stationary food particles. Each of these agents has a way to sense the environment and can act based on their perceptions. We compare the results when using different learning methods for the agents.

2 Background

2.1 Genetic algorithms

"Genetic Algorithms are search algorithms based on the mechanics of natural selection and natural genetics" Goldberg (1989). The main idea behind genetic algorithms is that there is a population of individuals, where each individual is represented by a genotype, which can be anything from bit strings, integer vectors, to the weights of a neural network. Each individual represents a solution to the problem, and the main goal of these methods is to find the best solution according to some criteria. For example, when maximising a function of two variables $f(x,y)$, each individual's genotype consists of 2 numbers, x and y.

Every individual is then assigned a *fitness* score, which measures how close the individual is to the desired solution. In the above example, the fitness value of an individual (x_i, y_i) could simply be $f(x_i, y_i)$.
This allows the highest fitness individuals to be chosen to reproduce which can be done in many ways. This reproduction method is called crossover, where the genotypes of two parents are combined to form the genotype of the child. Mutation is then performed on the child, which randomly alters its genotype to maintain genetic diversity and facilitate exploration. In the above example, breeding parents (x_i, y_i) and (x_j, y_j) can result in a child (x_j, y_i). Mutation simply changes some of the values, which would (for example) result in $(x_j + n, y_i), n \sim N(0,1)$.
Ultimately by adding individuals to the reproduction pool in proportion to their fitness, the highest fitness individuals reproduce more often. This results in good traits that perform well being maintained and ill-performing traits being discarded.

This procedure is then repeated many times until a sufficiently good solution is found.

2.2 Reinforcement Learning

Reinforcement learning is an area of machine learning where an agent learns through interaction with its environment and the feedback that it receives.

Reinforcement learning is usually performed on a Markov Decision Process (MDP) which consists of a set of states, a set of actions that can be performed in each state, a reward that is associated with each state and action pair and finally a transition function which determines the next state of a system given the current one and the action that was performed (Sutton and Barto, 2018). The goal of the agent is to try and achieve the maximum reward possible. One way to do this is to associate a value with each action and learn which actions will most likely give the greatest total reward over the course of the future.
To also allow for better results the agent sometimes chooses random actions instead of the predicted best action in an attempt to explore more of the action space. This prevents the agent from fixating on known actions and allows it to discover potentially superior actions.
This is known as the exploration vs exploitation trade-off and Sutton and Barto (2018) capture this nicely: "The agent has to exploit what it has already experienced to obtain rewards, but it also has to explore to make better action selections in the future."
The specific method used in this report was Q-learning, with linear function approximation, and we used the state directly as the features. An ε-greedy policy was used, which means that a random action was chosen with probability ε and a greedy action was chosen otherwise. For evaluation, we set $\varepsilon = 0$ to fully exploit the learned policy. We also used eligibility traces, to achieve a compromise between a one-step method and a full episode, Monte Carlo one.

The main idea behind Q-learning is to have a learned action value function, $Q(s,a)$, and we update Q using (Watkins, 1989).

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1},a) - Q(S_t,A_t)]$$

We use a slightly different formulation to account for eligibility traces and function approximation (Sutton and Barto, 2018, Chapter 12).

$$\begin{aligned} Q(S_t,A_t) &= w_t(A_t) \cdot S_t \\ \delta_t &= R_{t+1} + \gamma \max_a Q(S_{t+1},a) - Q(S_t,A_t) \\ z_t &= \gamma\lambda z_{t-1} + S_t \\ w_{t+1} &= w_t + \alpha z_t \delta_t \end{aligned}$$

2.3 Evolutionary Reinforcement Learning

Evolutionary reinforcement learning (ERL) is a method that attempts to combine reinforcement learning and evolutionary search to perform better than either method individually. The seminal work on this subject is the one by Ackley and Littman (1992), in which the authors describe agents as entities that have an action network that controls their behaviour and an evaluation network that controls how 'good' the agents perceive a certain state to be. This evaluation network provides the reinforcement learning signal to the action network. The evaluation network's weights are specified solely by the evolutionary process and are static during an agent's lifetime. The action network's initial weights are also passed down to the next generation, but the agents can modify these using reinforcement learning.

In our report, we simplify this slightly, in that the agents only have an action-choosing network and the reward signals are generated by the environment.

There has been other work performed on this topic, like Khadka and Tumer (2018) in which the authors use reinforcement learning to inject gradient information into the population of agents that are evolved using a genetic algorithm. They apply this method to standard reinforcement learning domains and show that in general, ERL performs better and converges faster on hard tasks than a pure genetic algorithm or reinforcement learning method.

3 Methods

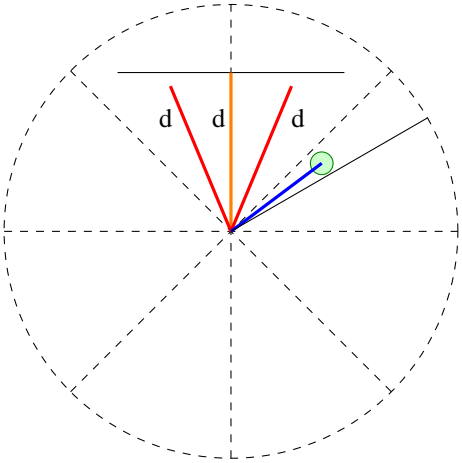
As described in Section 2, the methods we used were Q-learning, a genetic algorithm (GA) and a simple implementation of evolutionary reinforcement learning. We chose these for a few reasons:

- ERL is an attractive method to use, as it emulates the natural world's hierarchical improvement, using evolution and natural selection to pass information between generations, but each individual has the capacity to learn during their lifetime (Wang *et al.*, 2019). This method is of course much simpler than the natural world, but the concepts are somewhat valid.
- For ERL, we chose to use Q-learning as the reinforcement learning method, because it is simple to implement and intuitive. Q-learning (albeit the Deep Q Network algorithm) was also used by Wang *et al.* (2019) for a similar ecosystem.
- Since ERL is our main method, and it consists of a genetic algorithm and reinforcement learning, it makes sense to compare it to its constituent parts to determine if the combination of the two outperforms either one (or both).

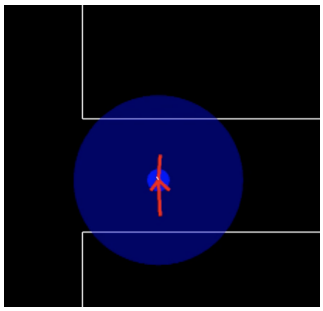
4 Simulation

We created a simple simulation that consists of stationary food particles, stationary obstacles, predators and prey. The rules of the simulation are as follows:

- When a prey touches a food particles, it eats the food, which disappears. The prey gains positive reward and some energy.



(a) The state as observed by an agent. The green circle is a piece of food, the straight line is a wall.



(b) A screenshot of the simulation. The agent is the blue dot, the walls are white and the red lines are the agent's observations.

- When a predator touches the prey, the prey dies and the predator gains energy and reward.
- When either a prey or predator touches an obstacle, they receive a negative reward and come to a stop.
- When a predator or prey has more energy than a specific threshold, they can reproduce which results in a mutated copy being created.
- When an agent's energy or lifetime reaches 0, they die.

Each agent can observe the world in a small radius V around it. The observations are represented as a 32 element vector, which is split up into 8 groups of 4. Each of these groups represents a sector of the visible circle (as portrayed in Figure 1a) and the four numbers are the values of $1-F/V, 1-P/V, 1-Pr/V, 1-W/V$, where F, P, Pr, W are the distances to the closest food particle, prey, predator and obstacle (wall) in that sector respectively. Each of these values is set equal to V when there is no such element in the visibility range, resulting in a 0 in the state at that position.

The actions of the agent at each step is simply a direction to apply an acceleration in (N, NE, E, etc). The agent's velocity is then modified by this acceleration and it is capped at some upper limit to ensure that the agents don't move too fast and erratically.

5 Data

This project did not require any explicit training data, as the agents learned from their interactions with the environment. Thus, the data was in fact tuples of the form $(s_t, a_t, r_{t+1}, s_{t+1})$, where s_t is the state at timestep t, a_t is the action the agent took and r_{t+1} is the reward the agent obtained from performing action a_t in state s_t. The agents used this to learn a better action-value function. The genetic algorithm did not use this information at all, since it only took into account the fitness of each individual to select which individuals to breed to form the next generation.

Our experiments generated a large amount of data that included the training performance of agents with different parameters and their evaluation results. For each experiment, there were also 5 nearly identical runs, where the only difference was the initial random seed. We analysed all of this data to generate plots and tables as well as to obtain insight into the mechanics and performance of each agent and parameter combination. We also stored the intermediate and final agents from each experiment to be able to perform evaluation again on the trained agent, or use it in unseen environments to investigate how it generalises.

6 Experiments

Our first experiment involves a simplified version of the above where there is only food and a single prey agent (the observations are adjusted accordingly). This prey never dies and also never reproduces. This simplified experiment was done to be able to compare different learning algorithms in a controlled environment. As described in Section 3, we used 2 different learning methods to act as a baseline for ERL, namely a Genetic algorithm and a simple linear Q learning implementation.
We trained each agent on the equivalent of 2500 episodes of 4000 timesteps each, which corresponds to $T = 10$ million timesteps in total. At the end of every episode, the simulation was reset to a different random state. For Q-learning, the agent simply experienced 2500 episodes and learned in each one.

For the genetic algorithm and ERL, we used $G = \lceil \frac{T}{4000P} \rceil$ generations of P individuals, which totals to roughly 10 million total evaluations (as at each generation, each individual in the population performs 4000 timesteps of the simulation).

The genetic algorithm and ERL implementation thus only saw G different episodes, as the training method used the same level on the whole population and only learned during that. So, ERL did not perform much learning but made up for that with a genetic algorithm.

We trained each agent and recorded the training rewards.

We then evaluated these agents on a set of hand-crafted, unseen evaluation levels, one containing only obstacles, the other only food and the last one containing both. We also evaluated the agents on random levels from the same distribution as the training levels. This was done to determine how the agents perform in each subtask (collecting food and avoiding obstacles), as well as how they perform overall.

The agent was run 100 times on each of these levels and we recorded the results.
Note, when doing the evaluation, we constrain the reward to not go below -2000. The reason for this is that occasionally, an agent got stuck in a corner, which then resulted in the agent obtaining very negative rewards (of the order of -1×10^5). Since the agent did not learn during evaluation, when facing a situation like this, the average of all the 100 runs is very skewed towards this outlier. Thus, when the agent receives a reward of less than -2000 during evaluation, we stop the episode there. This resulted in more balanced results, and the effect of a few outliers was minimised.

All of the above was repeated 5 times with a different random seed each time to attempt to control for random variations. We report the mean of these 5 seeds, and usually include the standard deviation as the shaded in parts of the graph.

6.1 Multi-Agent Simulation

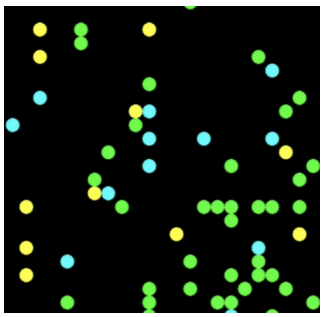


Figure 2: A screenshot of the multi-agent simulation. The green dots are food, cyan are prey and yellow are predators.

In our second experiment, we use the full multi-agent simulation, with different agents. We create a population of agents for each method (RL, GA, ERL) and just let the simulation run for some number of timesteps. We measure data such as the average energy per agent, how many agents there are of each learning method and each species (predator and prey). We then conclude which method performs the best.

Due to the high computational cost for the single-agent simulation, scaling this up to support multiple agents was infeasible. We instead discretised the environment into a grid, where each agent can occupy one grid cell. The visibility range of each agent was set to a neighbourhood of one cell around its current position. This enabled us to run much larger scale simulations with many agents.

Note, in many of the multi-agent results, we compare against a two baselines. The first is a random agent that simply performed a random action at each step. The second is a hard-coded agent that chooses actions based on custom rules that we created with the purpose of acting as a nearly optimal agent.

As initial, default hyperparameters (for both simulations), we use a population size of 50 for the genetic algorithm and ERL, and for ERL and Q-learning we use $\alpha = 0.005, \gamma = 0.99, \lambda = 0.9$. We also use an initial $\varepsilon = 0.1$ and decay it at every step by 0.999 (or 0.99 in the multi-agent case), until it reaches $\varepsilon_{min} = 0.02$. This is to facilitate more exploration at the beginning of the agent's lifetime, but choose actions more greedily as it learns.

These parameters where obtained from simple, small scale experimentation with initial values from Sutton and Barto (2018). We do investigate and attempt to optimise these parameters in the single-agent setting, but due to time and compute constraints we only use the default parameters in the multi-agent case.

This report focuses on the ERL method and we only show the results of the other methods as baselines. Thus, when optimising the parameters, we only consider ERL.

7 Results

Here we showcase the results obtained from our experiments. When plotting training rewards, we plot the 10 point moving average unless otherwise stated. We use some abbreviations when reporting the results for the sake of clarity. These are:

ERL Evolutionary reinforcement learning.

GA Genetic algorithm.

SimpleQ Q-learning.

Hard The hard-coded agents.

In Figure 3 we report the moving average of training rewards with a window size of 10. The shaded area is the standard deviation over the 5 random seeds. Table 1 and Figure 4 show evaluation results for each of the different agents and levels.

What we can see from Figure 3 is that all the methods managed to learn something as time went on. The ERL method has the highest reported training rewards, the genetic algorithm has the worst and SimpleQ's rewards are somewhere in between. The evaluation table tells a different story, however. SimpleQ performs the best, while ERL

Method	obstacles	foodhunt	combination	Fixed Random	Total
Random	-1936.0 ± 14.0	-1725.0 ± 51.0	-1992.0 ± 6.0	-1923.0 ± 9.0	-7576.0 ± 63.0
GA	-1062.0 ± 244.0	-569.0 ± 199.0	-1158.0 ± 251.0	-903.0 ± 267.0	-3691.0 ± 908.0
ERL	-384.0 ± 297.0	-14.0 ± 29.0	-143.0 ± 191.0	-163.0 ± 70.0	-703.0 ± 451.0
SimpleQ	-89.0 ± 81.0	1.0 ± 29.0	-495.0 ± 760.0	-93.0 ± 41.0	-676.0 ± 754.0
HardCode	-3.0 ± 2.0	34.0 ± 1.0	0.0 ± 0.0	18.0 ± 3.0	49.0 ± 3.0

Table 1: Evaluation Scores at the end. $\bar{x} \pm \sigma$

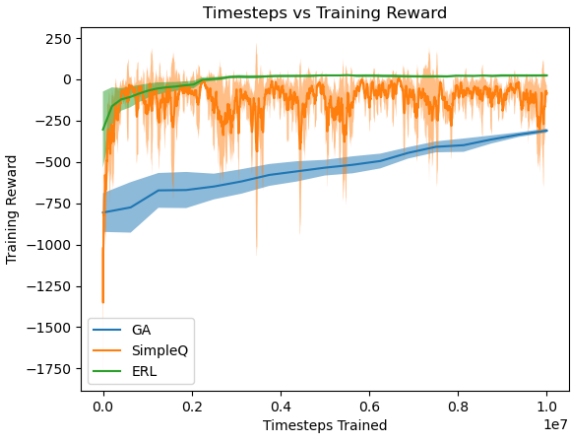


Figure 3: Training rewards.

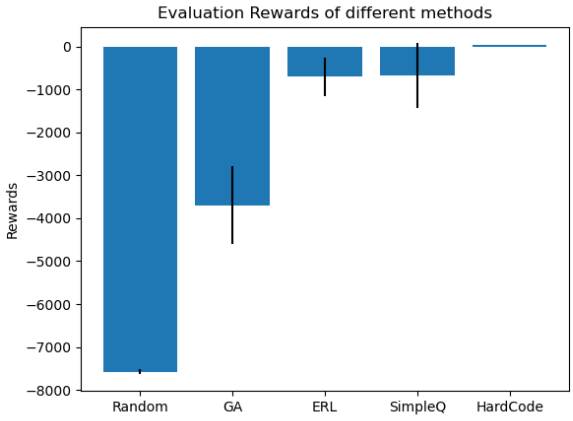


Figure 4: Evaluation Scores for each method.

performs slightly worse. All methods have a high variance, indicating that the random initialisation substantially affects the evaluation performance.

When looking at the actual visualisation of the agents in the evaluation levels, it does seem like the agents perform relatively well, but there are some cases where it gets stuck in a corner and performs badly. One reason for the big difference between training and evaluation rewards in the ERL case may be because we report only the best agent's training reward and not the average of the whole population, which is much lower. The best agent potentially overfits on the training levels, and doesn't generalise as well on other levels.

7.1 Optimisation

Here we attempt to optimise our ERL agent. The different techniques we will try are:

- Use a different population size.
- Use a different learning rate.
- Use different γ and λ parameters.
- Remove the eligibility traces to see how the agent performs without them.
- Add a bias term to the Q learning agent's network.
- See how early stopping affects the evaluation performance.

For each of these, we will run the experiments with a range of parameters and measure the rewards and evaluation scores for all of them.

7.2 Population Size

We first attempt to optimise the population size of the ERL method. We use multiple different population sizes but trained for the same amount of timesteps. Thus, when halving the population size, we double the number of generations. We experimented with the following values: $10, 25, 50, 100, 150$.

As we can see from Figure 5, the training rewards are relatively close, and they do vary somewhat. Table 2 shows that a population size of 50 (which resulted in 50 generations) performed the best. When using a very small population (of 10 say), the results are quite bad. This might be because there are not enough individuals to sufficiently explore the space.

For the next sections, we will continue to use a population size of 50.

Population Size	obstacles	foodhunt	combination	Fixed Random	Total
Population Size = 10	-736.0 ± 624.0	-86.0 ± 119.0	-562.0 ± 739.0	-372.0 ± 239.0	-1756.0 ± 1464.0
Population Size = 100	-511.0 ± 669.0	-20.0 ± 21.0	-569.0 ± 731.0	-411.0 ± 384.0	-1510.0 ± 1794.0
Population Size = 150	-250.0 ± 139.0	-28.0 ± 28.0	-385.0 ± 210.0	-391.0 ± 160.0	-1055.0 ± 306.0
Population Size = 25	-402.0 ± 299.0	1.0 ± 6.0	-111.0 ± 67.0	-427.0 ± 219.0	-939.0 ± 461.0
Population Size = 50	-384.0 ± 297.0	-14.0 ± 29.0	-143.0 ± 191.0	-163.0 ± 70.0	-703.0 ± 451.0

Table 2: The evaluation table for different population sizes of ERL

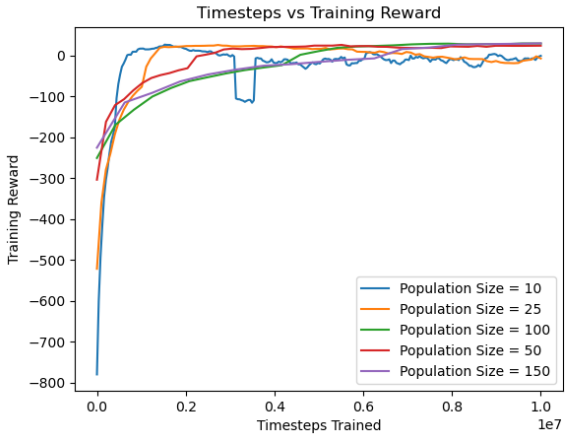


Figure 5: Training Rewards for different population sizes of ERL

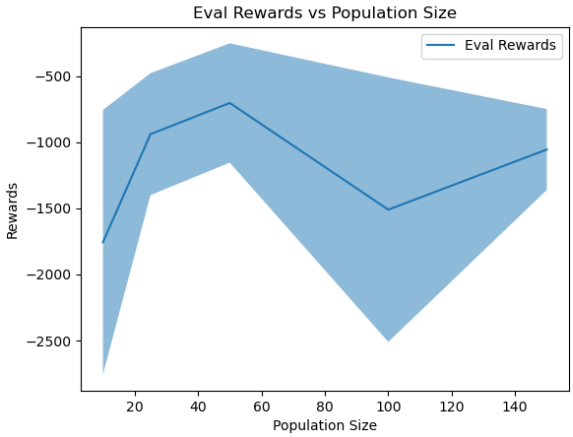


Figure 6: Total evaluation reward vs population size for the ERL method.

7.3 Hyperparameters

We perform a limited grid search over some parameters to attempt to understand how each parameter affects the learning capacity of the algorithm. We only choose a few options for each, as every training run took a long time and we had limited computing resources. We used a population size of 50, since that performed the best in the previous experiment.

In Table 3, we can see that the best results were the one that used $\alpha = 0.05, \gamma = 1, \lambda = 0$. Closely behind are other ones, with slightly smaller learning rates and non-zero λ. Since $\gamma = 1$, there is no reward discounting, so the agent values a reward now and a reward later the same. Using $\lambda = 0$ does not make use of eligibility traces and thus reduces to a 1 step TD method. This is quite surprising, as using a non-zero λ is expected to perform better (Sutton and Barto, 2018, Chapter 12).

7.4 Algorithm Parameters

Our final experiments for the single-agent system will be modifying the training length, as well as adding a bias to the ERL model.

7.4.1 Using a bias

In this experiment, we compare the results when using a bias term in our state, against the normal case, where we do not do this.

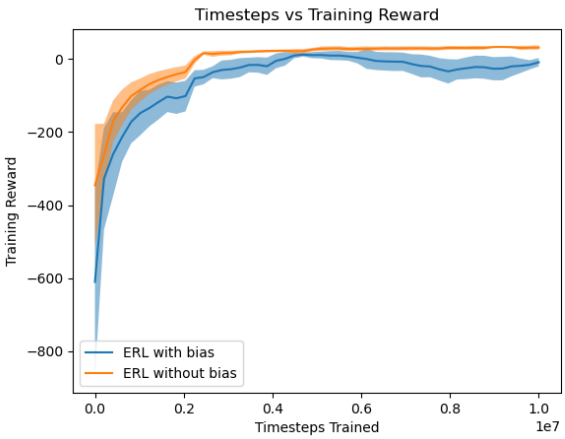


Figure 7: Comparing two runs with and without bias

We can see from Figure 7 and Table 4 that the run that did not use a bias vastly outperformed the one that did use a bias. This could be because of a few reasons, like that using a bias should not help the agent, as all actions are roughly equivalent without any extra state information. It could also

Parameters	foodhunt	combination	Fixed Random	Total
$\alpha = 0.0005, \gamma = 1, \lambda = 0$	-22.0 ± 31.0	-144.0 ± 135.0	-263.0 ± 162.0	-888.0 ± 623.0
$\alpha = 0.0005, \gamma = 0.99, \lambda = 0.95$	-73.0 ± 37.0	-193.0 ± 146.0	-371.0 ± 144.0	-885.0 ± 409.0
$\alpha = 0.005, \gamma = 1, \lambda = 0.95$	-30.0 ± 22.0	-20.0 ± 40.0	-361.0 ± 135.0	-872.0 ± 623.0
$\alpha = 0.005, \gamma = 0.99, \lambda = 0.9$	-14.0 ± 29.0	-143.0 ± 191.0	-163.0 ± 70.0	-703.0 ± 451.0
$\alpha = 0.05, \gamma = 1, \lambda = 0$	-12.0 ± 28.0	-216.0 ± 284.0	-293.0 ± 121.0	-694.0 ± 560.0

Table 3: Table of evaluaton results for all different hyperparameter combinations tested. This used $\varepsilon = 0$ for evaluation, and an ERL population size of 50. We only show the top 5 configurations for clarity

Bias or Not	obstacles	foodhunt	combination	Fixed Random	Total
ERL with bias	-440.0 ± 781.0	-86.0 ± 86.0	-474.0 ± 768.0	-430.0 ± 286.0	-1431.0 ± 1655.0
ERL without bias	-173.0 ± 195.0	-12.0 ± 28.0	-216.0 ± 284.0	-293.0 ± 121.0	-694.0 ± 560.0

Table 4: Comparing using a bias vs not using one. This used ERL with parameters $\alpha = 0.05, \lambda = 0, \gamma = 1$

make the Q learning more unstable or take longer to learn as there are more parameters.

7.4.2 Time trained

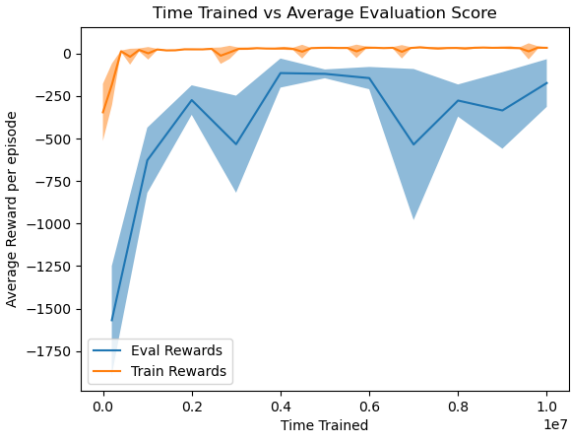


Figure 8: Evaluation score plotted vs time taken. No smoothing

In Figure 8 we show that the agent's evaluation score has improved after training for a while, but then it largely oscillated and worsened a bit. This worsening might indicate that the agents overfitted to the training domains. The same information is shown in Table 5, where we see that the agent performed badly initially, peaked at about 400000 steps and then gradually worsened.

7.5 Multi-Agent Simulation

In this experiment, we implement the multi-agent simulation. This is similar to the single-agent simulation, but the crucial difference is that multiple agents can exist and interact within the world. We have two types of agents, prey which simply eat food and predators that eat prey.

Each agent has an energy level, which decreases over time and gets replenished when they eat. Each agent also has a lifetime, and they die when the lifetime expires. Agents can reproduce when they have enough energy. Reproduction in this case is performed by simply copying the single parent agent, and mutating it slightly. This method is chosen to simulate the process of evolution more generally. Each agent can also learn during their lifetime, and these learned weights are passed down to the next generations. This is roughly the structure followed by Ackley and Littman (1992); Fraczkowski and Olsen (2014).
In this experiment, we only compare ERL with the random and hard-coded agents since ERL fits naturally into this learning and evolution framework.

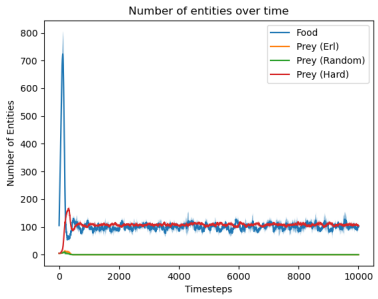
For this section, we measure the number of each species and how this evolves over time to determine the nature of the interactions between the agents.

We started off with different amounts of preys in separate populations (e.g. of a total of 30 preys, 10 were ERL agents, 10 were random and 10 were hard-coded agents). We then ran the simulation multiple times and observed the population sizes as time went on. For all of these experiments, we ran the same experiment multiple times (5 or 10) with different random seeds. We then removed the outliers with the least and most amount of agents respectively and report the average of the middle 3 or 8.

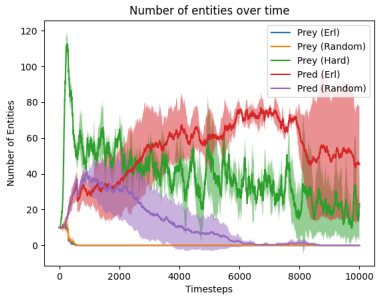
Figure 9a shows how the environment has a fixed carrying capacity, and the number of prey cannot go above that number for long. This makes sense, as we had a hard limit on how much food there could be in the simulation at any given time. In Figure 9b we see that the hard-coded prey triumphed over the other two populations, and there is a balance between predator and prey. At about timestep 6000, the random predators die out and the ERL predators increase in numbers.

Time Trained	obstacles	foodhunt	combination	Fixed Random	Total
2000000	-438.0 ± 255.0	-24.0 ± 30.0	-296.0 ± 262.0	-338.0 ± 92.0	-1095.0 ± 347.0
10000000	-173.0 ± 195.0	-12.0 ± 28.0	-216.0 ± 284.0	-293.0 ± 121.0	-694.0 ± 560.0
6000000	-195.0 ± 135.0	8.0 ± 17.0	-154.0 ± 88.0	-238.0 ± 111.0	-578.0 ± 262.0
5000000	-89.0 ± 66.0	-12.0 ± 24.0	-144.0 ± 75.0	-233.0 ± 67.0	-478.0 ± 103.0
4000000	-177.0 ± 209.0	2.0 ± 18.0	-105.0 ± 114.0	-179.0 ± 70.0	-460.0 ± 342.0

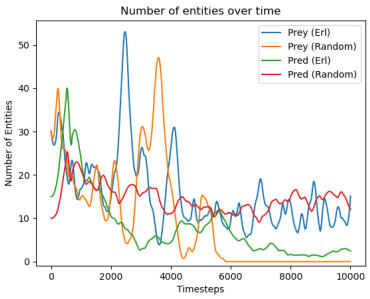
Table 5: Evaluation scores as a function of train time



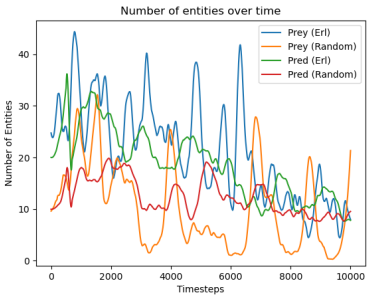
(a) Only the hard-coded agents survive.



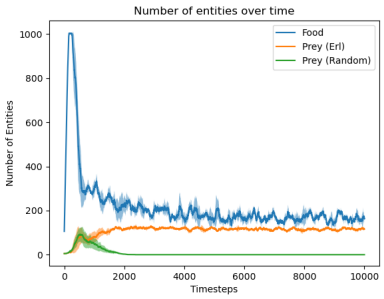
(b) Balance between predator and prey.



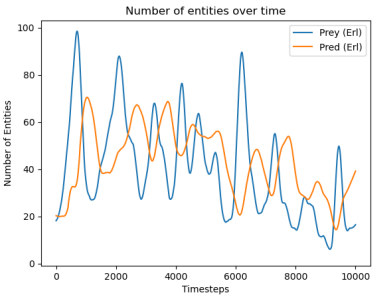
(c) Random prey die out.



(d) Competition between prey.



(e) The ERL agents eradicate the sub-par random agents.



(f) Cyclic nature of predator and prey.

Figure 9: Some multi-agent results. The X-axis is timesteps, Y-axis is the number of entities

In Figure 9c and Figure 9d we plotted the multi-agent results, smoothed with a window size of 201, with the errorbars are left off for clarity. We can see some interesting behaviour, for example in Figure 9d the growth of the ERL prey and Random prey are inversely correlated, i.e. when one increases the other decreases. This indicates that these two populations are competing for the same resources (food).

Figure 9c showed how the ERL agent outperformed the random agent, but the dominant predator was the random one. We found this happened frequently, i.e. either the predator was random and the prey ERL, or vice versa. This might indicate that stable equilibria only result when there is some gap in the abilities of the predators and prey.

In Figure 9e we can see that the ERL agents eventually outcompeted the random agents, causing these random agents to die out.
Finally, in Figure 9f (which is smoothed), we see that the predators and prey oscillate and that the oscillations are slightly out of phase. Thus, as predators increase in number, the number of prey decreases as they are being eaten. Because there are less prey, the number of predators decreases again due to lack of food. This periodic behaviour can be found in nature and the mathematical models thereof (Martin and Ruan, 2001; Takeuchi, 1996).

8 Discussion

This report focused on exploring the evolutionary reinforcement learning algorithm in an open-ended, ecosystem-based environment. We compared ERL to other methods, specifically a simple genetic algorithm and a Q-learning implementation. The training reward for ERL was the highest, which could be attributed to having multiple individuals, and the results that were reported were the maximum rewards, and the average was much lower. We evaluated each method on a collection of levels, and we found that Q learning and ERL performed similarly, and both had a large variance. As expected a Random agent performed much worse,

and a hand-crafted agent performed quite well.

We then attempted to optimise some of the hyperparameters of the ERL method, including the learning rate, discount factor (γ) and eligibility traces decay (λ). We also used different population sizes and found that an intermediate population size performed the best. Using a bias in the neural network caused performance to degrade and agents that were trained for 500000 steps were better than the ones that were trained for the whole 10000000 steps. This early stopping resulted in an agent that outperformed Q-learning and had a much smaller variance in evaluation reward. We thus had success in optimising our initial hyperparameters.

The multi-agent system gave us a testing bed to compare methods more organically, and we found that the ERL method has the potential to learn and survive in an environment, but it was still outperformed by the hand-crafted agents. These hand-crafted agents are not easily generalisable though, as for every new domain/simulation one has to implement the new rules from scratch, where a method like ERL can easily be applied to many domains without any change (except for the number of input and output neurons). ERL outperformed the random agents in general, showing that some cleverness is needed to succeed in this environment. We performed some interesting visualisations and data-visualisation at the end of the project.

Our Q-learning implementation (and also the RL component of the ERL method) used eligibility traces, as explained in (Sutton and Barto, 2018, Chapter 12). The authors did discuss some potential issues when using Q-learning with eligibility traces and indicated that Watkin's $Q(\lambda)$ method (Watkins, 1989) addresses some of the concerns. The main difference in this method is setting the traces to 0 whenever a non-greedy action was taken. We did not use this technique, and using it might improve the results for both Q-learning and ERL.

9 Conclusion and Future Work

Our evolutionary reinforcement learning method performed on par with Q learning, and for the best hyperparameters, even surpassed it slightly. Both of these methods still performed much worse than the hard-coded one. The variance in our evaluation reward might indicate that the agents overfitted to some degree, as the training reward was quite high.

The multi-agent simulation gave us some interesting insights into the behaviour in terms of competition of our agents in a more natural setting. As expected, our method lies somewhere between random and hard-coded.

We did identify some limitations, namely that the variance in evaluation reward was quite large, indicating that randomness played a large part in the performance of the agents. The multi-agent simulation was also somewhat affected by randomness, and even the random agents survived often, by simply randomly exploring the environment.
Future work can be to adapt this to use other reinforcement learning methods, or apply it to other, less open ended domains. Another direction is to modify the multi-agent system to make it less lenient to random actions, and ensure that agents that survive are required to be somewhat more intelligent than random agents.

In conclusion, we implemented a basic ERL agent, compared it against others and investigated the effect of hyperparameters and training time. We analysed the results and found that ERL can perform well, and survive in a competitive multi-agent simulation.

The code for the agents, simulation and visualisation is available in our Github repository[1].

10 Acknowledgements

Computations were performed using High Performance Computing infrastructure provided by the Mathematical Sciences Support unit at the University of the Witwatersrand.

References

[Ackley and Littman 1992] Ackley and Littman. Evolutionary reinforcement learning (erl). 1992.

[Fraczkowski and Olsen 2014] Rachel Fraczkowski and Megan Olsen. An agent-based predator-prey model with reinforcement learning. In *Conference: Swarmfest 2014: 18th Annual Meeting on Agent-Based Modeling & Simulation*, 06 2014.

[Goldberg 1989] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*, chapter 1. Addison-Wesley Longman Publishing Co., Inc., USA, 1st edition, 1989.

[Huse *et al.* 1999] Geir Huse, Espen Strand, and Jarl Giske. Implementing behaviour in individual-based models using neural networks and genetic algorithms. *Evolutionary Ecology*, 13:469–483, 01 1999.

[Khadka and Tumer 2018] Shauharda Khadka and Kagan Tumer. Evolutionary reinforcement learning. *arXiv preprint arXiv:1805.07917*, 2018.

[Martin and Ruan 2001] Annik Martin and Shigui Ruan. Predator-prey models with delay and prey harvesting. *Journal of Mathematical Biology*, 43(3):247–267, 2001.

[1] https://github.com/Michael-Beukman/ERLEcosystemSimulation

[Sunehag *et al.* 2019] Peter Sunehag, Guy Lever, Siqi Liu, Josh Merel, Nicolas Heess, Joel Z Leibo, Edward Hughes, Tom Eccles, and Thore Graepel. Reinforcement learning agents acquire flocking and symbiotic behaviour in simulated ecosystems. In *Artificial life conference proceedings*, pages 103–110. MIT Press, 2019.

[Sutton and Barto 2018] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[Takeuchi 1996] Yasuhiro Takeuchi. *Global dynamical properties of Lotka-Volterra systems*, chapter 2. World Scientific, 1996.

[Wang *et al.* 2019] Xueting Wang, Jun Cheng, and Lei Wang. Deep-reinforcement learning-based co-evolution in a predator–prey system. *Entropy*, 21(8):773, 2019.

[Watkins 1989] Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. 1989.