A Comparison of Algorithms Playing EvoMan

1st Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

2nd Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

3rd Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

***Abstract*—This paper describes a comparison between algorithms for evolving agents for the game Evoman. We have tried a cascade ensemble method, starting with an algorithm that is either fast either has an explorative bias and refining its solution with an exploitative algorithm. The algorithms we have tested for the first stage are Q-learning, genetic algorithms and particle swarm optimization. All of these algorithms are searching for the weights of a neural network of a fixed structure. Both the genetic algorithms and the particle swarm optimization algorithms were tested with a sparse and an iterative approach. The best explorative algorithm was the sparse particle swarm optimization. The exploitative algorithm we tested is proximal policy optimization. Using PPO with random initialization lead to better results than an ensemble method of PSO cascaded with PPO. We also detected overfitting of the agent trained with PPO.**

***Index Terms*—game-playing agent, artificial intelligence, EvoMan, genetic algorithm, reinforcement learning, q-learning, neuroevolution, particle swarm optimization, proximal policy optimization, ppo**

I. Introduction

This paper presents our solution for the ”Evoman: Game-playing Competition for WCCI 2020” [1]. We are attempting to train an agent playing the 2D shooting game Evoman [2]. We have tried an ensemble cascade method with two stages. For the first stage we tested algorithms that either found acceptable solutions fast either had a high explorative bias. The algorithms we considered for this stage are Q-Learning [6], genetic algorithms [7] and particle swarm optimization [8]. Both the GA and the PSO train the weights of a neural network of a fixed structure. Both the GA and the PSO were tested with a sparse approach and an iterative approach. For the exploitative stage we used Proximal Policy Optimization [10].

II. Problem Description

A. Environment

EvoMan [2], [3] is a framework for testing competitive game-playing agents. This framework is inspired by Mega Man II [5], the game created by Capcom. EvoMan is a 2D shooting game where the player controls an agent playing against an opponent. The agent will collect information about the environment through 20 sensors (Fig 1):

- 16 correspond to horizontal and vertical distances to a maximum of 8 different opponent projectiles.

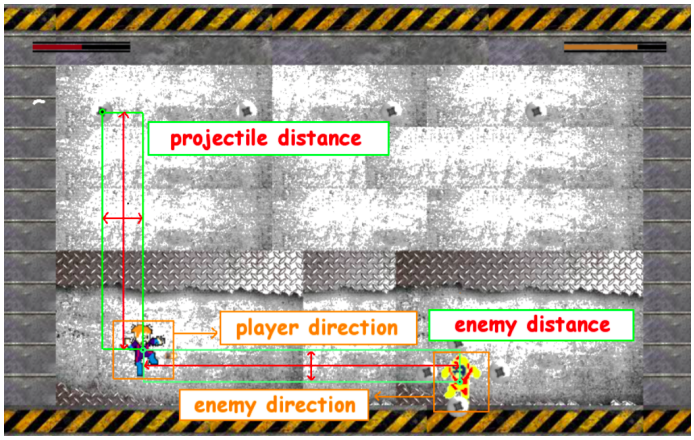

Fig. 1. Sensors available for the player [2].

- 2 correspond to the horizontal and vertical distance to the enemy.
- 2 describe the directions the player and the enemy is facing

The actions which the agent may take are:

- walk left
- walk right
- jump
- shoot
- release of the jump

The lives of the player and the enemy start at 100. Everytime one of them gets hit, their life deplenishes. Whoever’s life reaches 0 loses the game.

In the original Capcom game the player would have to beat 8 opponents and acquire their weapons as they are defeated. The additional difficulty of EvoMan comes from the fact that the player has to defeat all the opponents using only the starting weapon. Each opponent can be fought on a specified difficulty level. The difficulty level is an integer greater or equal than 1 which is translated into a factor for the damage taken and damage given by the player, the higher the difficulty level the lower the damage given and the higher the damage taken. The framework is freely available[1] and it is currently

[1]https://github.com/karinemiras/evoman_framework

compatible with Python 3.6 and 3.7. There is also an extensive documentation available[2].

B. Problem

Beating an opponent is relatively easy when using specialized models against a specific enemy. The authors of the EvoMan framework trained multiple specialized agents against all opponents. After playing a game, the formula for the gain by which an agent is evaluated is:

$$gain = 100.01 + player\_life - enemy\_life$$

The highest final gain is 185.67 [2] and it was obtained with the NEAT [9] algorithm.

The problem we are trying to solve is a generalization of the one above. We are to train an agent against four enemies and then test its performance against all eight. The final score metric of an agent is the harmonic mean of the gains against all enemies. The combination of four enemies we are to train the agent is not fixed. The difficulty level for solving this problem is 2 for all enemies in training and testing, but we did some exploratory work with greater level of difficulties outside training and testing for obtaining the final solution for the problem.

The specifications above, like the number of enemies chosen for training or the difficulty level, come directly from a competition organized by the creators of the framework [1].

III. Approach

Our intention was to develop a cascade ensemble method where the resulting model of an algorithm which can achieve a decent gain fast or has a high exploratory bias is the starting point of an algorithm with high exploitative bias. The algorithm we expected to obtain a decent gain fast is a classic Q-learning [6] algorithm. The exploratory algorithms we have considered are genetic algorithms [7] and particle swarm optimization [8]. The exploitative algorithm we used was Proximal Policy Optimization [10].

To find the best bootstraping algorithm we have made the following changes, which had no impact in the training and testing for the final solution to the problem:

- we have increased the difficulty from the default 2 to 5.
- the evaluation is made only on the second opponent due to the varied environment.
- the gain function was modified as in Karine Miras' analysis [11] to:

$$0.9 \cdot (100 - enemy\_life) + 0.1 \cdot player\_life - ln(nr\_of\_game\_timesteps)$$

The best exploratory algorithm would be trained on four enemies and used in cascade with proximal policy optimization without the changes mentioned in this section.

[2]https://github.com/karinemiras/evoman_framework/blob/master/evoman1.0-doc.pdf

A. Q-Learning

The algorithm we expected to obtain a decent gain fast is a classic Q-learning [6] algorithm with neural networks.

We used a neural network to predict the reward function for each possible move (left, right, shoot, jump, release of jump) from a given state. We then take the action with the highest predicted reward. The input of this neural network is composed of the current game sensors and the previous 2 game sensors with the moves taken at that specific point.

The neural network used for experiments has 2 hidden layers with 32 neurons each. Each layer has l2 regularization applied with a weight decay of 0.01 and sigmoid activation. After each predicted move, we updated the neural network using backpropagation, using as input the game sensors and as output the true reward function. A game ends when either the agent or the enemy lose all life. We have trained the agent on 5000 games. The average number of frames per game for the best model is 287.

B. Evolving Neural Network Weights with Genetic Algorithms

We used genetic algorithms to evolve the weights of neural networks with fixed structures.

1) Sparse Reward Genetic Algorithm: The second algorithm is a sparse reward neuroevolution [4]. The reward is defined as "sparse" because an agent doesn't find out how well it's doing until the end of the game, with no feedback during the game. An individual is represented as the weights of a neural network.

The neural network role is to predict the next move using the current game sensors and the previous 2 game sensors with the moves taken at that specific point. We used 2 hidden layers with 32 neurons for each individual.

We start with randomly initialized neural network weights and then represent them as a bitstring. The weights used for the neural network were values between 2 and -2 with a precision of 6 digits.

For evaluation, the bitstring is transformed into the weights of the neural network and a game is played, from which we can obtain the individual fitness.

Since we are representing the individuals as bitstrings we were able to apply a simple genetic algorithm [7].

The configurations we have used for the genetic algorithm is:

- population size: 50
- number of generation: 500
- crossover rate: 0.7
- mutation rate: {0.008, 0.1}
- elitism: 1

We have tried two experiments with different mutation rates in order to observe whether a very high mutation rate can lead to good results for the problem.

The solution of the genetic algorithm is the best individual from the last generation. Since we have used an elitism of 1, this means that the solution of the genetic algorithm is the best individual ever evaluated.

2) Iterative Genetic Algorithm: The next algorithm is an iterative neuroevolution. It is "iterative" because the agents are trained on a small number of game steps first and then the number of game steps slowly increases.

After a number of generations we increase the number of game timesteps the agents are allowed to train on. The fitness function was scaled based on the number of game timesteps in a way that if an agent training on x game timesteps will always have a fitness lower than an agent training on y game timesteps if x is lower than y.

C. Searching Neural Network Weights with Particle Swarm Optimization

We have used particle swarm optimization [8] to search for the weights of a neural network of a fixed structure which maximize the fitness defined in this section. We used the same neural network configuration as in the case of genetic algorithms.

The configurations used for the particle swarm optimization [8] algorithm are:

- population size: 30
- number of iterations: 200
- cognitive weight: {0.4, 0.8, 1.5}
- social weight: {0.8, 0.4, 3}
- inertia weight: {constant 1, decreasing from 1 by 0.0035 every epoch}

The same separation between an sparse and an iterative search was made in the case of particle swarm optimization, as in the case of the genetic algorithms.

We also tried to search with pso using an uniformly decreasing inertia weight with respect to the percent of the iterations passed, having values from 1 to 0.3. The inertia weight controls the level of exploration versus exploitation, higher values in the beginning of the algorithm lead to better exploration while lower values towards the end of the algorithm lead to better exploitation. This was tried to make the PSO focus even more on exploitation, rather than exploration.

D. Proximal Policy Optimization

PPO [10] is the algorithm we have used for exploitation. For this stage we have used the default difficulty of 2, the training was done on 4 opponents and the evaluation was done as described in the Problem Description(II).

The neural network is randomly initialized.
The configuration we have used for the PPO is:

- hidden layers: (64, 64)
- steps per epoch: 10000
- epochs: 3000
- gamma: 0.99
- clip_ratio: 0.2
- pi_lr: 3e-4
- vf_lr: 1e-3
- train_pi_iterations: 80
- train_v_iterations: 80
- lambda: 0.97
- target_KL: 0.01

E. Particle Swarm Optimization Cascaded With Proximal Policy Optimization

The output weights of a neural network resulted from a PSO search is the starting agent for PPO. In order to solve the generalized problem we have made the following changes from the PSO from the exploratory stage:

- The sizes of the hidden layers were increased from (32, 32) to (64, 64).
- It is trained on four opponents, not only one.
- The fitness function is the one from the Problem Description (II).

The same configuration was used for PPO as in the case of the PPO with random weights initialization.

IV. Experimental Investigation

A. Q-Learning

The Q-learning algorithm was the starting point of our comparison. Even when trained and evaluated against the same opponent, it would lose every game while inflicting almost no damage.

B. Genetic Algorithms

The iterative genetic algorithm leads to much better results than Q-learning (Fig. 2).

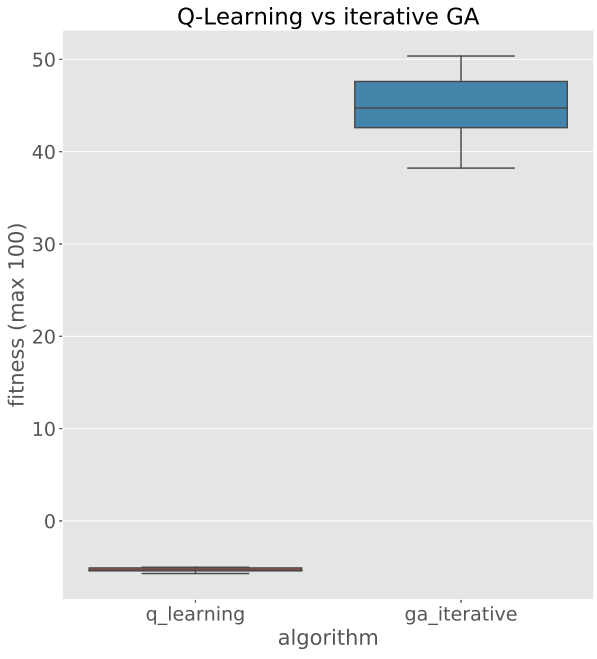

Fig. 2. The fitness comparison between Q-learning and iterative genetic algorithms.

We have shown that a mutation rate of 0.008 leads to better results than a mutation rate of 0.1 (Fig. 3).

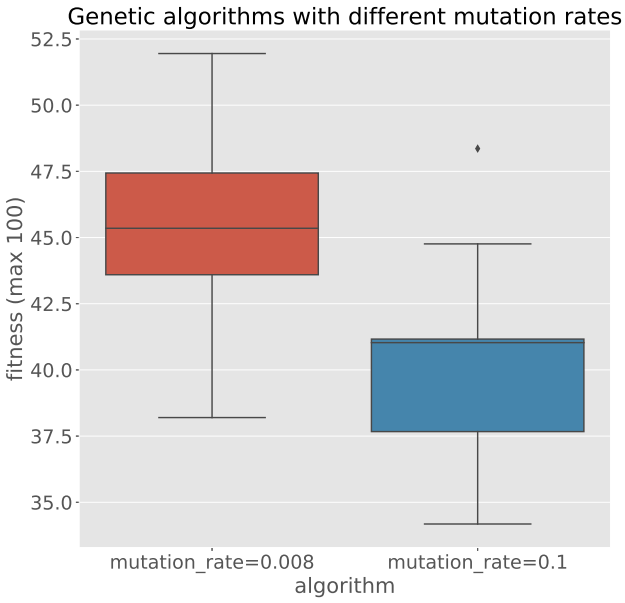

Fig. 3. The fitness comparison between sparse genetic algorithms with mutation rate of 0.1 and 0.008.

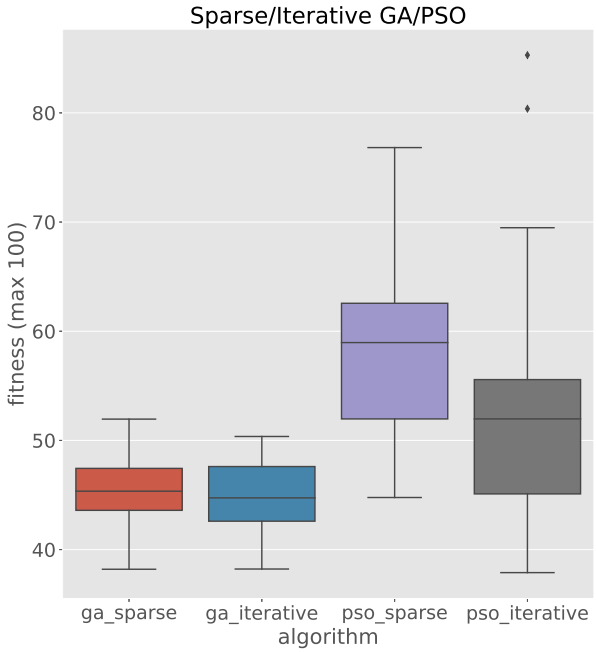

Fig. 4. The fitness comparison between the iterative and the sparse approached, both for the genetic algorithms and the PSO.

The results of the sparse genetic algorithm and the iterative genetic algorithm are not significantly different (Fig. 4).

C. Particle Swarm Optimization

Both particle swarm optimization algorithms lead to much better results than either of the sparse and iterative genetic algorithms. The iterative PSO lead to worse results than the sparse PSO. In Fig. 4 we can see that the best exploratory algorithm is the sparse PSO.

The time advantage of iterative PSO vs sparse PSO is not significant (Fig. 5).

We have also searched for good PSO weights for our problem. In the next stages we have used a cognitive weight of 0.4 and a social weight of 0.8 due to its higher variance in results (Fig. 6).

We have also tried two PSO inertia weight updates (Fig. 7). We continued with the constant inertia weight approach.

After deciding what the starting model should be, we have tested its performance (Fig. 8) and run time (Fig. 9) on multiple difficulties.

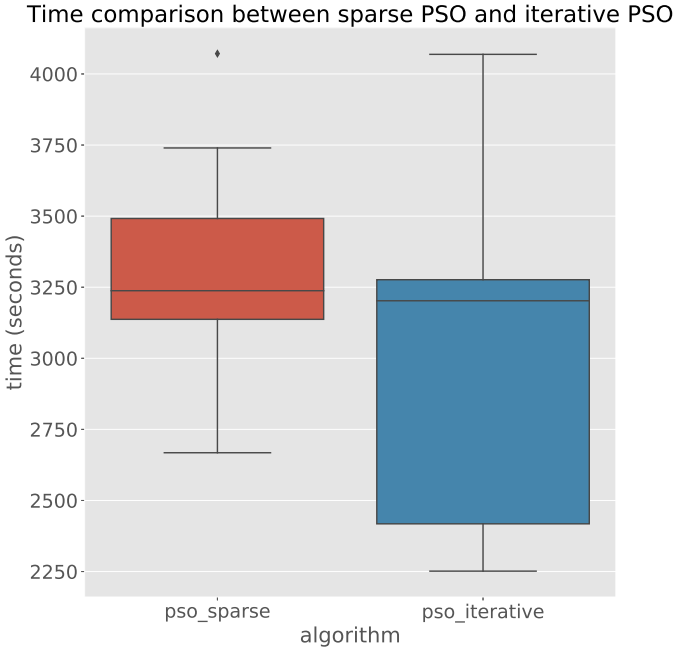

Fig. 5. The time comparison between sparse and iterative PSO.

V. Results

For evaluation of an agent, we ran 30 games against each opponent, leading to 8 averages (one per opponent), of which we computed the harmonic mean which is the final result of the agent.

A. Best Combination of Opponents for Training

We searched for the opponents that lead to the best results. The first combination of train opponents we considered was {1, 2, 6, 7} which was chosen empirically after manually playing against every opponent and chosing the ones which

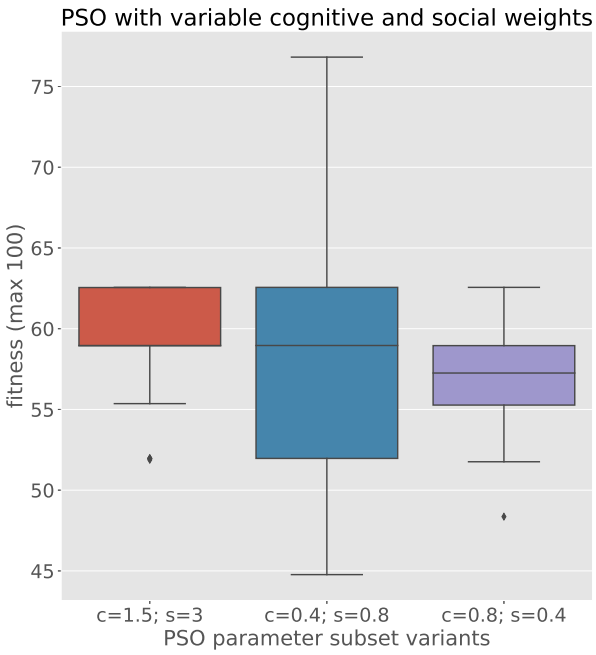

Fig. 6. The fitness comparison between sparse PSO with different cognitive and social weights.

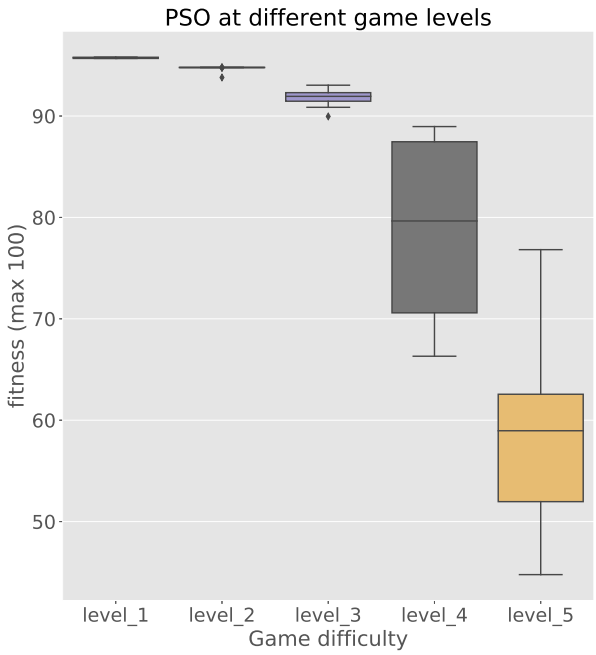

Fig. 8. The fitness comparison of PSO against different game difficulty levels.

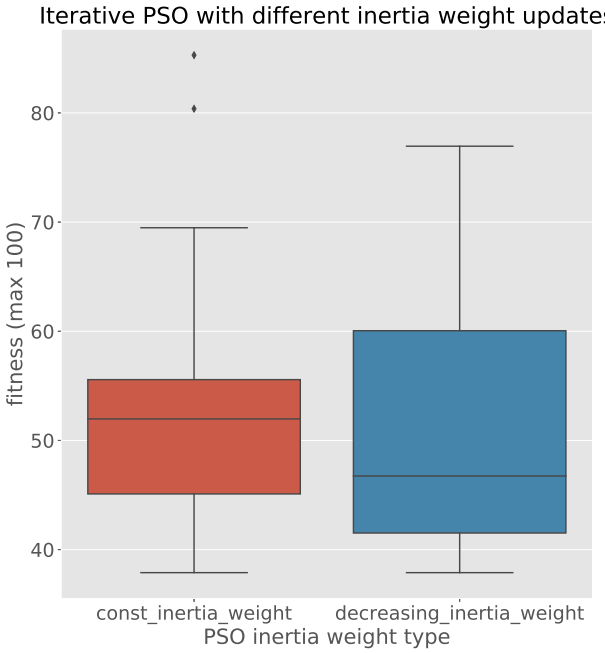

Fig. 7. The fitness comparison between PSO with constant inertia weight and with uniformly decreasing in time inertia weight.

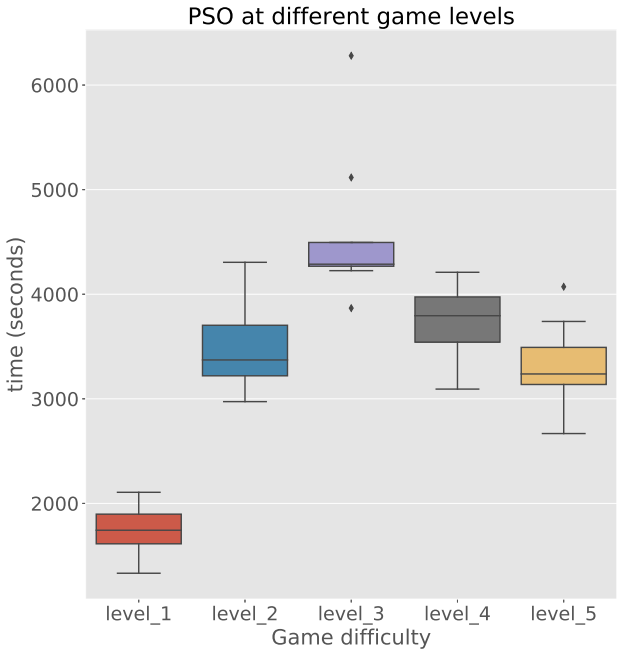

Fig. 9. The time comparison of PSO at different game difficulty levels.

behaved in such a way that covered the most general concepts our agents could learn in our opinion. We observed its final

gain as 125.37. We realized a set of exploratory experiments where we would search for other combinations of train opponents, but starting with the resulting model of PPO trained for 1000 epochs on the combination of opponents {1, 2, 6, 7}. If by starting with a pre-trained model another combination of train enemies does not lead to better results, then we can believe that the respective combination does not lead to better results than {1, 2, 6, 7}.

TABLE I
RESULTS FOR VARIOUS OPPONENTS WHEN STARTING WITH A PRE-TRAINED PPO MODEL ON ENEMIES {1, 2, 6, 7}

Train opponents	gain (harmonic mean)
$\{1,2,6,7\} \cup \{3,4,5,8\}$	44.16
$\{1,2,6,7\} \cup \{3,4,6,7\}$	75.3
$\{1,2,6,7\} \cup \{1,3,6,7\}$	110.19
$\{1,2,6,7\} \cup \{2,3,6,7\}$	136.1
$\{1,2,6,7\} \cup \{2,4,6,7\}$	85.22

The only combination that lead to a better score than 125.37 was {2, 3, 6, 7}. We ran an experiment with PPO with random initialization with the combination of train enemies {2, 3, 6, 7} and we obtained a final gain of 99.97. This means that we did not find a better combination of train opponents than {1, 2, 6, 7}, so it is the one we used for the following experiments.

B. Random Initialization PPO vs PSO Cascading PPO

The best PSO configuration was used in cascade before the PPO in 4 runs. PPO with random initialization was ran 3 times. The small number of runs is due to the long run time of the training. The worst result of the PPO with random initialization is higher than the best result of PSO cascaded with PPO (Fig. 10). Our explanation for getting worse results when using the technique mentioned above is that the function landscape is very big and PSO doesn't explore enough. Another possible explanation would be that the landscape is tricky when it comes to searching for generalist versus specialized solutions. We could try solving this problem by giving more exploratory power to PSO, but we did not experiment with this due to the lack of time.

C. PPO on the best configuration

Considering the results above, we decided that PPO with random initialization and enemies {1, 2, 6, 7} chosen for training would be the best configuration. We ran 3000 epochs of this configuration, saving all the models along the way after every 250 epochs. After looking at the testing results, we noticed that the final gain is greater after 2000 epochs than after 3000 epochs (Fig. 11). This means that there is overfitting during the training. We can conclude that by stopping the PPO algorithm after 2000 epochs instead of 3000 epochs we can end up with more generalized agents that perform better on average against all opponents.

D. Best Train Agent vs Best Tested Agent

The best train gain was obtained in the first run of the PPO with random initialization. The train gain was computed as

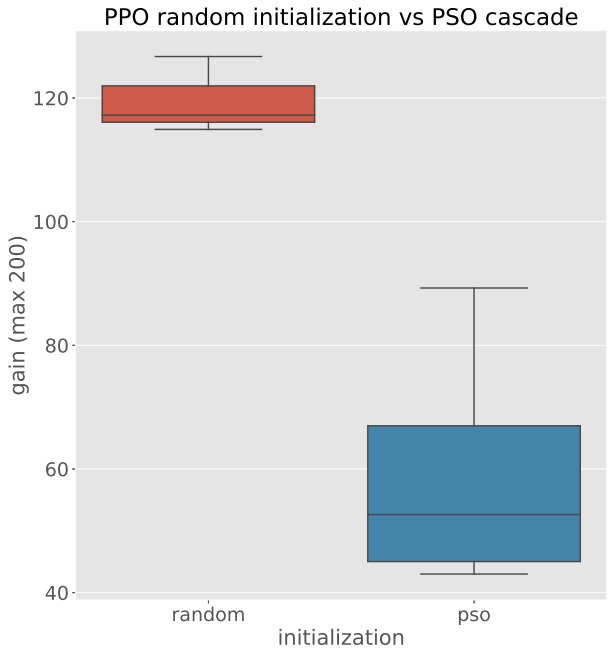

Fig. 10. Gain comparison between PPO with random initialization and with PSO cascading.

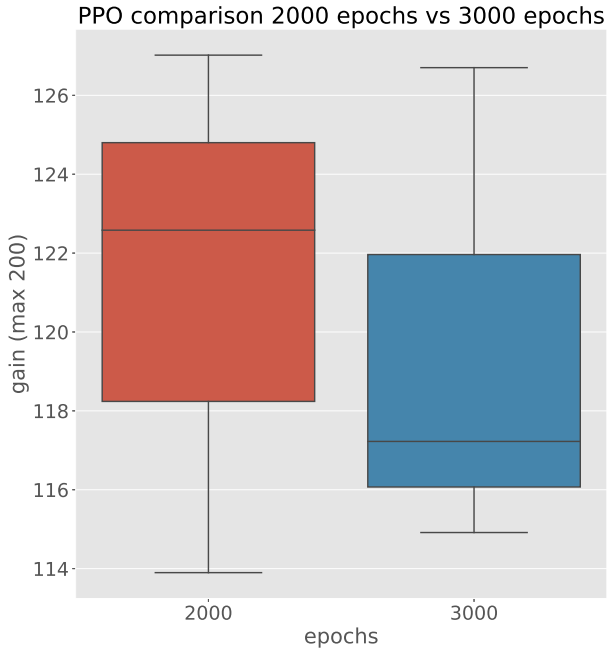

Fig. 11. Gain comparison between PPO after 2000 epochs and after 3000 epochs.

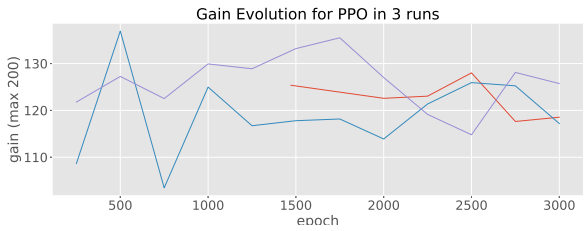

Fig. 12. Gain Evolution for PPO in 3 runs, each model was ran 30 times on each opponent for calculating the gains

the harmonic mean of the averages per opponent of the 30 games played against the enemies {1, 2, 6, 7}. The highest train gain is 198.4. When computed against all opponents, this agent has an overall gain of 117.22. Out of the three PPO with random initialization runs, the highest final gain is 127.01. After the experiments were concluded we also looked at the overall gains of the agents before 3000 epochs (Fig. 12). For the runs 2 and 3 the snapshots of the models were saved every 250 epochs. In the first run these periodic snapshots start after 2000 epochs. When looking at the results of the intermediary agents we observed that there are some high peaks in test gain. The highest test gain was observed during the third run after 1750 epochs and it has a value of 137.18.

E. Comparison with the Upper Bound

With the PPO algorithm trained on the opponents {1, 2, 6, 7} we have beat the best specialized models from the original paper which were given as upper bounds [2].

TABLE II
SPECIALIZED NEAT VS GENERALIZED PPO (GAIN)

Opponent	**PPO**				**Specialized** NEAT
	Run 1 3000 ep (best at training)	***Run 2*** 3000 ep	***Run 3*** 3000 ep	***Run 3*** 1750 ep (best tested)	
1	198.41	199.07	199.81	199.61	190.01
2	199.74	199.27	190.34	189.14	194.01
3	58.94	46.67	70.27	85.01	180.01
4	58.34	66.5	64.11	80.17	194.01
5	172.61	165.65	174.61	158.83	194.01
6	195.73	196.95	195.85	194.37	173.01
7	199.79	195.05	193.27	191.17	177.01
8	122.17	145.49	141.89	140.61	186.01
{1, 2, 6, 7} harmonic mean	198.4	197.57	194.75	193.49	183.09
final harmonic mean	117.22	114.91	127.01	137.18	185.67

F. Specialized PPO

In order to obtain a score higher than the upper bound we have tried two methods:

- A generalized PPO agent trained against all 8 opponents.
- A specialized PPO agent for each of the 8 opponents.

TABLE III
SPECIALIZED NEAT VS GENERALIZED PPO (PERCENTAGE OF GAMES WON)

Opponent	**PPO**			
	Run 1 3000 ep (best at training)	***Run 2*** 3000 ep	***Run 3*** 3000 ep	***Run 3*** 1750 ep (best tested)
1	100	100	100	100
2	100	100	100	100
3	3.33	0	3.33	46.66
4	0	6.66	0	6.66
5	100	96.66	100	100
6	100	100	100	100
7	100	93.33	100	100
8	80	100	96.66	90

TABLE IV
AVERAGE PLAYER AND ENEMY LIFE. AVERAGE GAME DURATION

Opponent	**Best train agent**			**Bes**	
	Avg player life	***Avg enemy life***	***Avg duration***	***Avg player life***	***Av***
1	97.26	0	159.63	100	
2	98.33	0	176.6	89.73	
3	0.13	43.33	457.26	7.66	
4	0	37.66	547.5	1.84	
5	71.32	0	379.13	60.52	
6	96.32	0	165.06	95	
7	99.74	0	162.06	85.9	
8	34.3	6	520.93	38.76	
Average	62.17	10.87	321.02	59.92	

1) PPO trained against all opponents: At the default difficulty of 2, a PPO agent trained against all opponents obtained a final gain of 185.57, being lower with 0.1 than the upper bound found with specialized NEAT. For such an agent we have also studied how its gain changes when the difficulty is modified.

TABLE V
PPO TRAINED AGAINST ALL OPPONENTS (GAIN)

Opponent	**Difficulty**				
	1	***2***	***3***	***4***	***5***
1	200.01	199.54	170.75	155.48	41.08
2	200.01	200.01	182.81	168.81	168.51
3	199.38	194.81	164.31	127.54	130.61
4	186.14	180.61	154.23	181.58	60.28
5	196.56	187.77	185.34	142.41	158.31
6	196.35	174.43	179.19	23.84	9.88
7	199.26	183.07	95.40	8.01	8.28
8	196.57	169.31	123.00	41.34	10.01
harmonic mean	196.68	185.57	149.57	35.76	20.89

2) A PPO agent for each opponent: In the second attempt to outscore the upper bound we have trained an agent for each of the 8 opponents. Due to the high computational time this experiment was done only at difficulty 2 and the number of training epochs for PPO was decreased from 3000 to 1000. It won all its games against all opponents at difficulty 2.

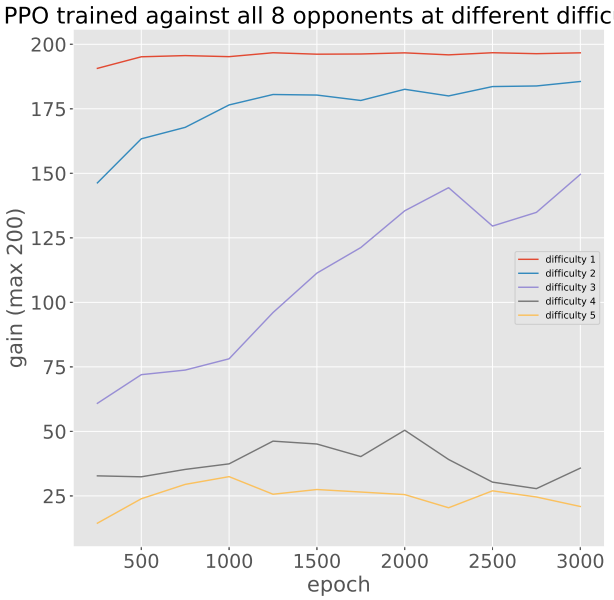

Fig. 13. Gain Evolution for PPO when trained against all 8 opponents.

TABLE VI
PPO TRAINED AGAINST ALL OPPONENTS (PERCENT GAMES LOST)

Opponent	Difficulty				
	1	*2*	*3*	*4*	*5*
1	0	0	20	23.33	100
2	0	0	0	0	0
3	0	0	0	13.33	3.33
4	0	0	0	0	60
5	0	0	0	0	0
6	0	0	0	100	100
7	0	0	56.66	100	100
8	0	0	10	100	100

TABLE VII
PPO GENERALIZED VS PPO SPECIALIZED VS NEAT SPECIALIZED

Opponent	PPO g.	PPO s.	NEAT s.
1	199.54	199.67	190.01
2	200.01	199.61	194.01
3	194.81	199.94	180.01
4	180.61	195.95	194.01
5	187.77	198.03	194.01
6	174.43	199.67	173.01
7	183.07	197.73	177.01
8	169.31	195.07	186.01
harmonic mean	185.57	198.19	185.67

G. Specialized PPO at difficulty 5

A PPO agent for each opponent. 1000 training epochs per opponent. (10 million frames for each opponent). Harmonic mean of gains is 154.58.

TABLE VIII
SPECIALIZED PPO AT DIFFICULTY 5

Opponent	Gain	Percent games lost
1	166.01	13.33
2	197.18	0.00
3	72.28	100.00
4	191.23	0.00
5	196.71	0.00
6	198.76	0.00
7	176.01	0.00
8	172.61	0.00

H. Time Analysis

Since the match time is relevant, the machine on which the games were run is also relevant. The tests were run on an i7 4750HQ processor. The training was done on 3 threads. All computations were done on the CPU. The median game time during training is 28 seconds. The average number of frames per game is 287.

I. Specialized PPO at multiple difficulties

TABLE IX
SPECIALIZED PPO AGAINST MULTIPLE DIFFICULTIES (GAIN)

Opponent	Difficulty			
	2	*3*	*4*	*5*
1	199.68	196.51	189.21	166.01
2	199.61	199.91	199.21	197.18
3	199.94	199.51	147.21	72.28
4	195.95	196.14	198.81	191.23
5	198.03	193.38	191.73	196.71
6	199.67	195.18	196.85	198.76
7	197.73	192.45	184.85	176.01
8	195.07	185.37	178.05	172.61
harmonic mean	198.19	194.7	184.12	154.58

TABLE X
SPECIALIZED PPO AGAINST MULTIPLE DIFFICULTIES (PERCENTAGE GAMES LOST)

Opponent	Difficulty			
	2	*3*	*4*	*5*
1	0.00	0.00	0.00	13.33
2	0.00	0.00	0.00	0.00
3	0.00	0.00	36.67	100.00
4	0.00	0.00	0.00	0.00
5	0.00	0.00	0.00	0.00
6	0.00	0.00	0.00	0.00
7	0.00	0.00	0.00	0.00
8	0.00	0.00	0.00	0.00

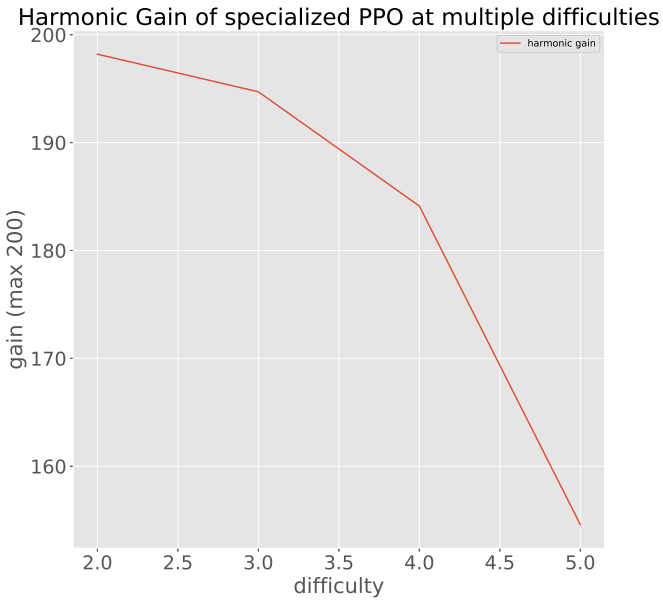

Fig. 14. Harmonic Gain of specialized PPO at multiple difficulties.

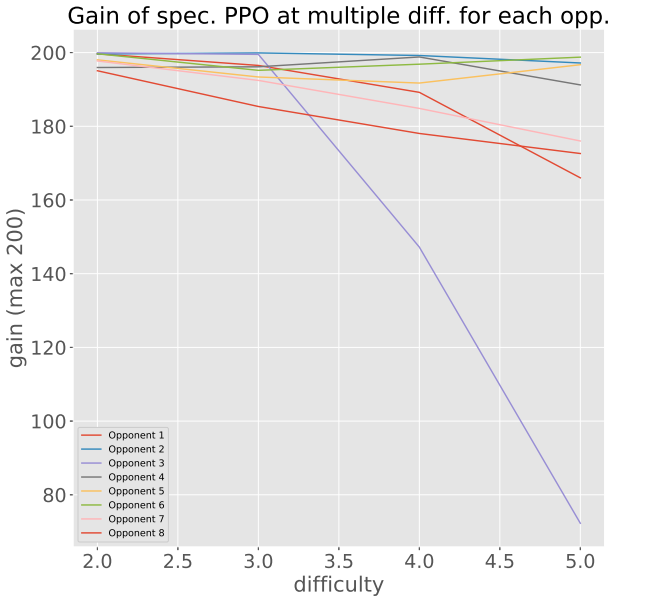

Fig. 15. Gain of specialized PPO at multiple difficulties for each opponent.

VI. Conclusions

After observing that the sparse methods were better than the iterative ones, PPO with random initialization is better than PSO cascaded with PPO and PPO with random initialization trained on four opponents leads to better train results than specialized agents trained with NEAT we can conclude that the problem space is in such a way that the methods with a greedy bias are moving away from the global optimum.

References

[1] Evoman: Game-playing Competition for WCCI 2020, http://pesquisa.ufabc.edu.br/hal/Evoman.html

[2] Fabricio Olivetti de Franca, Denis Fantinato, Karine Miras, A.E. Eiben and Patricia A. Vargas. "EvoMan: Game-playing Competition" arXiv:1912.10445

[3] de Araújo, Karine da Silva Miras, and Fabrício Olivetti de França. "An electronic-game framework for evaluating coevolutionary algorithms." arXiv:1604.00644 (2016).

[4] Floreano, D., Dürr, P. & Mattiussi, C. Neuroevolution: from architectures to learning. Evol. Intel. 1, 47–62 (2008). https://doi.org/10.1007/s12065-007-0002-4

[5] M. MEGA, "Produced by capcom, distributed by capcom, 1987," System: NES.

[6] Watkins, C.J.C.H., Dayan, P. Q-learning. Machine Learning 8, 279–292 (1992)

[7] Holland J.H., Genetic Algorithms and Adaptation. Adaptive Control of Ill-Defined Systems, 1984, Volume 16 ISBN 978-1-4684-8943-9

[8] Kennedy, J.; Eberhart, R. (1995). "Particle Swarm Optimization". Proceedings of IEEE International Conference on Neural Networks. IV. pp. 1942–1948.

[9] Kenneth O. Stanly; Rist Miikkulainen (2002). "Evolving Neural Networks through Augmenting Topologies". Evolutionary Computation, Volume 10, Issue 2, Summer 2002, p.99-127, https://doi.org/10.1162/106365602320169811

[10] John Schulman, Filip Wolski, Prafulla Dhariwal, Alex Radford, Oleg Klimov (2017) "Proximal Policy Optimization Algorithms", arXiv:1707.06347v2

[11] Karine Miras, Evoman, https://karinemirasblog.wordpress.com/portfolio/evoman/