Increasing the Upper Bound for the EvoMan Game Competition

Gabriel-Codrin **Cojocaru**
Faculty of Computer Science
"Al. I. Cuza" University
Iasi, Romania
gabriel.cojocaru@info.uaic.ro

Sergiu-Andrei **Dinu**
Faculty of Computer Science
"Al. I. Cuza" University
Iasi, Romania
sergiu.dinu@info.uaic.ro

Eugen **Croitoru**
Dept. of Computer Science
"Al. I. Cuza" University
Iasi, Romania
eugennc@uaic.ro

***Abstract*—This paper describes a comparison between algorithms for evolving agents able to play the game Evoman. Our team took part in the "Evoman: Game-playing Competition for WCCI 2020", and won second place; beyond finding a good agent to satisfy the requirements of the competition - which aim at a good ability to generalise -, we have surpassed the existing non-general, best-known upper-bound. We have managed to exceed this upper bound with a Proximal Policy Optimization algorithm, by discarding the competition requirements to generalise. We also present our other exploratory attempts: Q-learning, Genetic Algorithms, Particle Swarm Optimisation, and their PPO hybridizations. Finally, we map the behaviour of our algorithm in the space of game difficulty, generating plausible extensions to the existing upper-bound.**

***Index Terms*—game-playing agent, Artificial Intelligence, EvoMan, Genetic Algorithm, Reinforcement Learning, Q-learning, Neuroevolution, Particle Swarm Optimization, Proximal Policy Optimization**

I. Introduction

This paper contains our improvement on the upper-bound for playing the 2D shooting / platformer game Evoman [3]. Our team took part in the "Evoman: Game-playing Competition for WCCI 2020" [1]. While placing second [2], we have managed to exceed the best-known upper-bound. This difference is due to the competition requirements for generalisation; the upper-bound has been provided by the organisers by dropping such requirements, and only by doing the same could we exceed it.

Our initial exploratory attempts focused on an ensemble, 2-stage cascade method. The first stage, designed to find good candidate solutions quickly, was either a Q-Learning [7], a Genetic Algorithm [8] or Particle Swarm Optimization [9] algorithm. The candidate solutions are weights on a fixed-structure Artificial Neural Network, which plays the Evoman game. The second stage, aimed at further refining the solutions found by the first stage, is a slower Proximal Policy Optimization [11] algorithm.

However, a simple random initialisation outperforms any of the first-stage algorithms, and the later experiments use PPO alone.

The competition required the algorithm to train with half the opponents in the game, and to test with all of them. Our method shows significant overfit in this challenge. However, its otherwise good performance makes it appropriate for setting a new upper bound, since the upper bound is being computed individually, by training and testing against each opponent in the Evoman game.

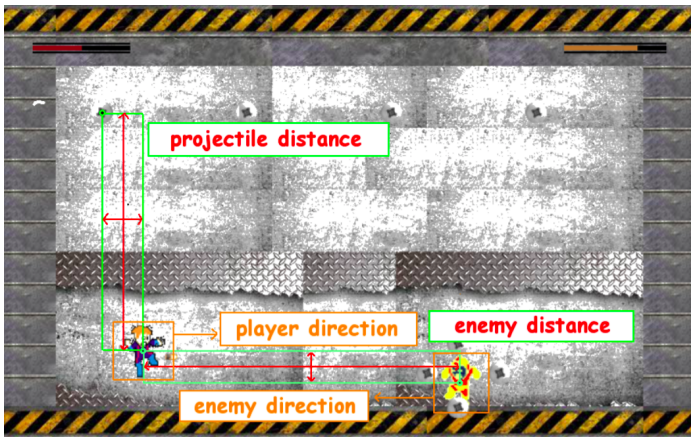

Fig. 1. Sensors available for the player [3].

II. Problem Description

A. Environment

Evoman [3], [4] is a framework for testing competitive game-playing agents. This framework is inspired by Mega Man II [6], a 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.
- 2 correspond to the horizontal and vertical distance to the enemy.
- 2 describe the directions the player and the enemy is facing

The 5 actions which the agent may take are: walk left, walk right, shoot, press jump, release jump.

The player and opponent start with 100 life points. On each hit received, their life points decrease. The player wins by reducing the opponent's life points to 0, and lose if their life points are reduced to 0 first.

In the original Capcom game, the player would have to beat 8 opponents and acquire their weapons once 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 at a specific 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]. There is also an extensive documentation available[2].

B. Problem

How well an agent performs against an enemy is measured by how much damage they inflict, and how much damage to themselves they avoid - or, conversely, how much life points they have remaining, and how much life points the opponent has at the end of the game:

$$gain = 100.01 + player\_life - enemy\_life \qquad (1)$$

As both start with 100 life points, the best possible value for gain is 200.01, for a player which defeats the opponent without getting hit.

The best known gain was 185.67 (as a harmonic average of 8 values, each for one of the 8 opponents) [3] and was obtained with a NEAT [10] algorithm. It is this value we've tried to exceed.

There are also various levels of generalisation available; the least is training a single agent on a single opponent, as the above method does. The next is training a single agent on all opponents: instead of having 8 specialised models for each opponent, a single unified model, trained on all 8, could be attempted. Furthermore, we could train a single agent on a subset of the 8, and test that agent on the full 8. The competition [1] allowed free choice of 4 training agents, and testing against all 8.

Another level of generalisation is learning from one difficulty level and applying to the others. While seeing this possibility, we did not attempt such a task, due to the current high difficulty of the problem.

III. Approach

Our initial intention was to develop a cascade ensemble method where a fast algorithm could provide a better starting point to a slower algorithm. We expected the classic Q-learning [7] algorithm to quickly obtain a decent gain. Two more exploratory algorithms we have considered are Genetic Algorithms [8] and Particle Swarm Optimisation [9]. The slower algorithm, intended to exploit the starting points provided by the first algorithm in the cascade, is a Proximal Policy Optimization [11] algorithm.

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

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

To find the best first-stage algorithm we have made the following changes (which had no impact in the training and testing for the final solution to the problem, other than guiding the choice of algorithm):

- we have increased the difficulty from the default 2 up 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 [15] to:

$$fitness = 0.9 \cdot (100 - enemy\_life) + 0.1 \cdot player\_life - ln(nr\_of\_game\_timesteps) \qquad (2)$$

The best possible value for the fitness is 100.

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

A. Solution structure

In all cases, a candidate solution is a fixed-structure feed-forward Artificial Neural Network.

The input and output layers are determined by the game framework. Each game frame gives us 20 sensor values, and we also remember the past 2 frames, alongside the movements we've made. Thus, $3 \cdot 20 + 2 = 62$ inputs.

We've decided on two hidden layers. At first, we've used 32 perceptrons on each, but later extended to 64 (only when using PPO). We haven't increased the number of perceptrons or hidden layers further, since our main issue proved to be overfit, not the ANN's inability to properly approximate the required function [13]. The activation function used is the Logistic Sigmoid; each layer has l2 regularisation applied to it, and each weight has a decay of 0.01. The weights are initialised in the range $[-2, 2]$, with a minimum precision of 6 decimal paces.

B. Q-Learning

We used a classic Q-Learning ANN to predict the reward function for each possible move. After each predicted move, we update the neural network using backpropagation. We compute the reward as the difference between the current score and the previous score:

$(prev\_enemy\_life - curr\_enemy\_life) \cdot 0.8 + (curr\_player\_life - prev\_player\_life) \cdot 0.2$

We have trained the agent on 5000 games.

C. Sparse Reward Genetic Algorithm

The second algorithm is a sparse reward Neuroevolution [5] algorithm. 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 represents the binary encoding of the weights of a Neural Network.

Since we are representing the individuals as bitstrings we were able to apply a Simple Genetic Algorithm [8].

The configurations we have used for the GA are:

- population size: 50
- number of generations: 500
- crossover rate: 0.7
- mutation rate: $\{0.008, 0.1\}$
- elitism: 1 individual

We have tried 2 experiments with different mutation rates in order to observe if a high mutation rate can lead to good results for the problem, since the role of this GA was chiefly exploratory in our ensemble.

The solution of the GA is the best individual from the last generation.

D. Iterative Genetic Algorithm

The next algorithm is an iterative neuroevolution one. 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. We've used this approach in order to save time; we believe it is reasonable to expect the initial agents to perform poorly, and thus require less time to decide on their quality.

The fitness function is scaled based on the number of game timesteps in a way that an agent training on fewer timesteps will always have a fitness lower than an agent training on more timesteps.

E. Particle Swarm Optimization

The configurations used for the PSO 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 iteration$\}$

The same separation between sparse and iterative evaluations was made in the case of PSO, as was in the case of GA.

F. Proximal Policy Optimization

We have devoted significant computational resources to a PPO [11], expecting it to exploit and refine solutions. The configuration we have used for the PPO is:

- randomly initialized weights
- 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

G. Particle Swarm Optimization Cascaded With Proximal Policy Optimization

The output weight configuration of an Artificial Neural Network resulting from a PSO search is the starting agent for PPO. While using the same PSO configuration as before, the ANN hidden layers size is increased from 32 to 64.

IV. Experimental Investigation

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

The iterative GA leads to much better results than Q-learning (Fig. 2). We've also chosen a mutation probability of 0.008. The results of the sparse GA and the iterative GA are not significantly different (Fig. 3).

Both PSO algorithms lead to much better results than either GA. The iterative PSO leads to worse results than the sparse PSO. In Fig.3 we can see that the best first-stage algorithm is the sparse PSO.

We have tested PSO performance (Fig. 4), on multiple difficulties.

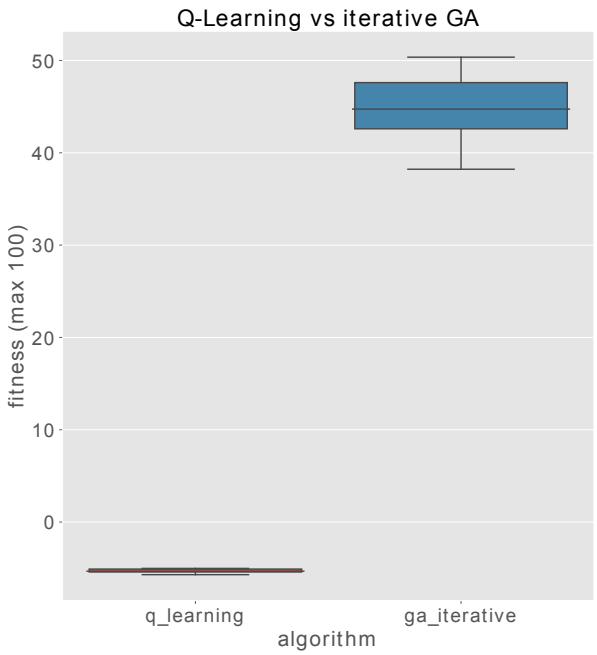

Fig. 2. The fitness comparison between Q-learning and iterative genetic algorithms. Difficulty: 5.

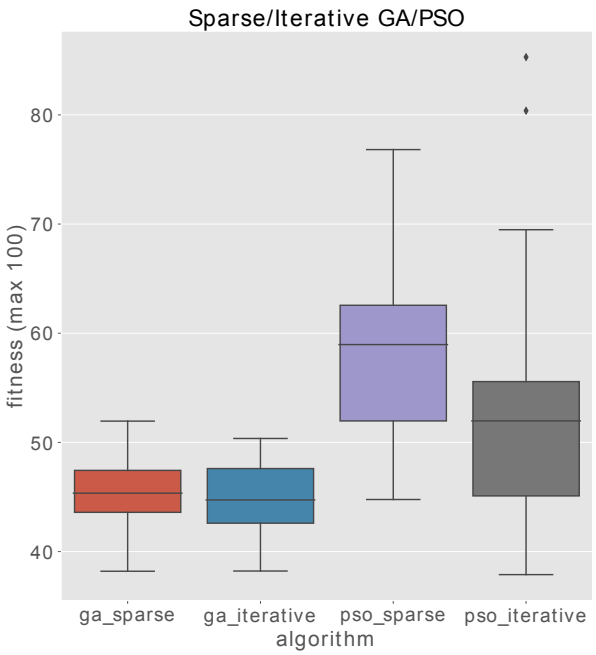

Fig. 3. The fitness comparison between the iterative and the sparse approached, both for the genetic algorithms and the PSO. Difficulty: 5.

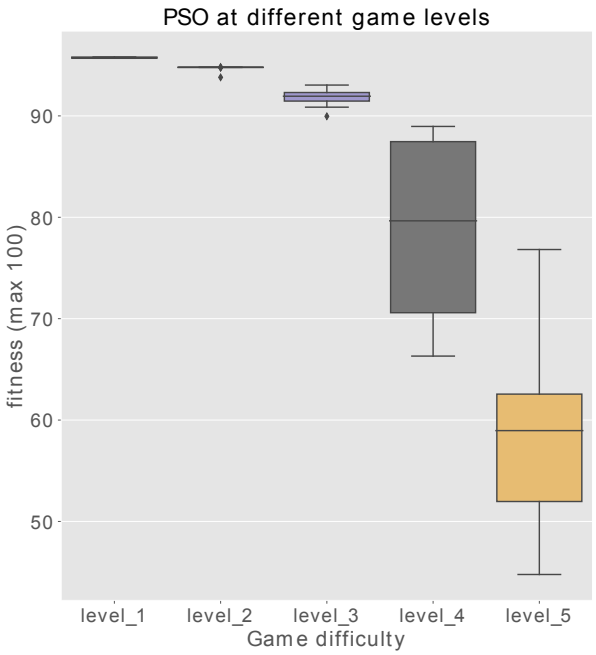

Fig. 4. The fitness comparison of PSO against different game difficulty levels.

V. Initial Results

For evaluating an agent, we ran 30 games against each opponent, leading to 8 averages (1 per opponent), of which we computed the harmonic mean, which is the final result for the agent. We use the harmonic mean because it is used for reporting other results for this problem [1].

A. Best Combination of Opponents for Training

We searched for the best 4 opponents to serve as a basis for generalization. The first subset considered was $\{1, 2, 6, 7\}$, which was chosen empirically after manually playing against every opponent. We choose the ones which exhibited the most general behaviors our agents could learn, in our opinion. We performed a non-exhaustive search, and found no other 4-opponent subset better than the human-chosen one.

B. Random Initialization PPO vs Cascading PSO + 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 training time. The worst result of the PPO with random initialization is far better than the best result of PSO cascaded with PPO, and we used this large difference to guide our choice, despite the small sample size. (Fig. 5).

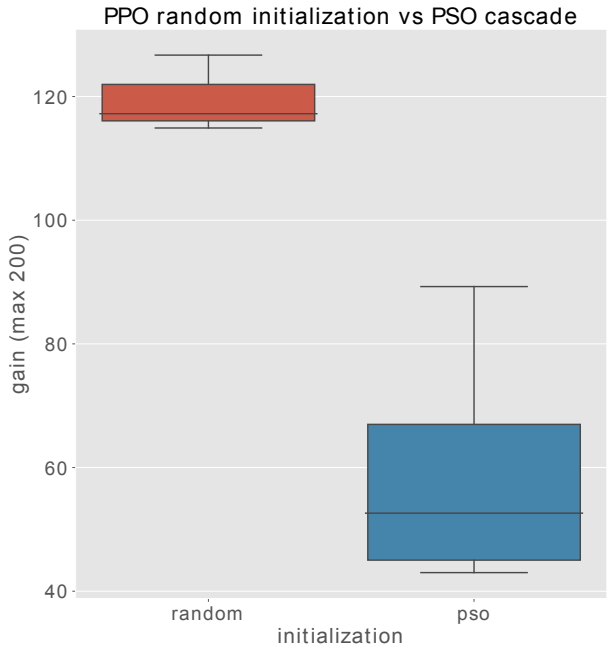

Fig. 5. Gain comparison between PPO with random initialization and with PSO cascading.

C. PPO in the best configuration

Considering the results above, we decided that PPO with random initialization and training enemies $\{1, 2, 6, 7\}$ is the

best configuration. We ran 3000 epochs (chosen *a priori*, considering only our experiences with PPO and general ANNs in other problems) in this configuration, saving all the models along the way after every 250 epochs - since we anticipated overfit. *After* looking at the testing results, we noticed that the final *test* gain is greater after 2000 epochs than after 3000 epochs (Fig.6).

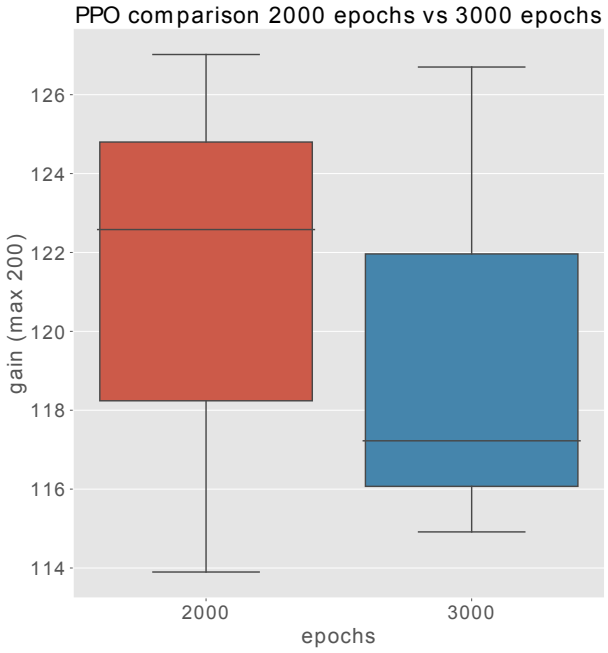

Fig. 6. Gain comparison between PPO after 2000 epochs and after 3000 epochs.

D. Best Train Agent vs Best Tested Agent

The highest training gain is 198.4. When computed against all opponents, this agent has an overall gain of 117.22. Out of the 3 PPO with random initialization runs, the highest test gain is 127.01 (but they had lower training gains, due to overfit). After the experiments were concluded, we also looked at the overall gains of the agents before 3000 epochs. The highest test gain was observed during the third run after 1750 epochs, and it has a value of 137.18.

We were considerably computation-starved in these experiments; requiring a minimum sample size of 30 meant we could only try out 3 different agents. The competition organisers, upon running our algorithm, reported a different (and higher) value than in our experiments, of 138.14 [2].

VI. Increasing the Upper Bound

Our PPO implementation wasn't able to generalise sufficiently, neither in the competition (where our gain of 138.14 was below the best gain of 146.88) [2], nor in our own tests.

TABLE I
SPECIALIZED NEAT VS GENERALIZED PPO (GAIN)

Opponent	**PPO**				**Specialized** NEAT
	Run 1 3000 ep best train	***Run 2*** 3000 ep	***Run 3*** 3000 ep	***Run 3*** 1750 ep best test	
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
$\{1, 2, 6, 7\}$	198.4	197.57	194.75	193.49	183.09
$\{1, \ldots, 8\}$	117.22	114.91	127.01	137.18	185.67

TABLE II
SPECIALIZED NEAT VS GENERALIZED PPO (PERCENTAGE OF GAMES LOST)

Opponent	**PPO**			
	Run 1 3000 ep best train	***Run 2*** 3000 ep	***Run 3*** 3000 ep	***Run 3*** 1750 ep best test
1	0	0	0	0
2	0	0	0	0
3	96.67	100	96.67	53.3
4	100	93.3	100	93.3
5	0	3.3	0	0
6	0	0	0	0
7	0	6.67	0	0
8	20	0	3.3	10

Overfitting was a significant problem, and we believe it to be a good indicator of our issues.

However, with the PPO trained on opponents $\{1, 2, 6, 7\}$, we have surpassed the best specialised opponent-specific models from the original paper, which were given as upper bounds [3]. Thus, we can understand this issue as an exploration versus exploitation trade-off: while we haven't been able to develop the best exploratory, generalising model, we seem to have found a very good exploitative, targeted solution.

As such, we've performed 3 further experiments which:

- develop a unified model for opponents $\{1, \ldots, 8\}$, training on opponents $\{1, \ldots, 8\}$, at difficulty 2.
- develop a unified model for opponents $\{1, \ldots, 8\}$, training on opponents $\{1, \ldots, 8\}$, at difficulties $\{1, \ldots, 5\}$.
- develop individual models for each opponent, training on each respective opponent, at difficulty 2.

We shall call the unified model "generalised" and the individual model(s) "specialised".

Beyond these changes in aim, we haven't modified the above-described PPO algorithm employed.

A. Single difficulty

The specialised PPO was able to surpass the specialised NEAT [3] algorithm, both in harmonic mean and in each individual case (Table III). The specialised PPO agents were trained for 1000 epochs, as they achieved excellent results.

TABLE III
PPO GENERALISED VS PPO SPECIALISED VS NEAT SPECIALISED, DIFFICULTY = 2

Opponent	PPO gen.	PPO spec.	NEAT spec.
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

Additionally, PPO was able to create a general model which is extremely close (a 0.1 gain difference) in harmonic mean to the results of the 8 specialised models evolved through NEAT.

B. Multiple difficulties

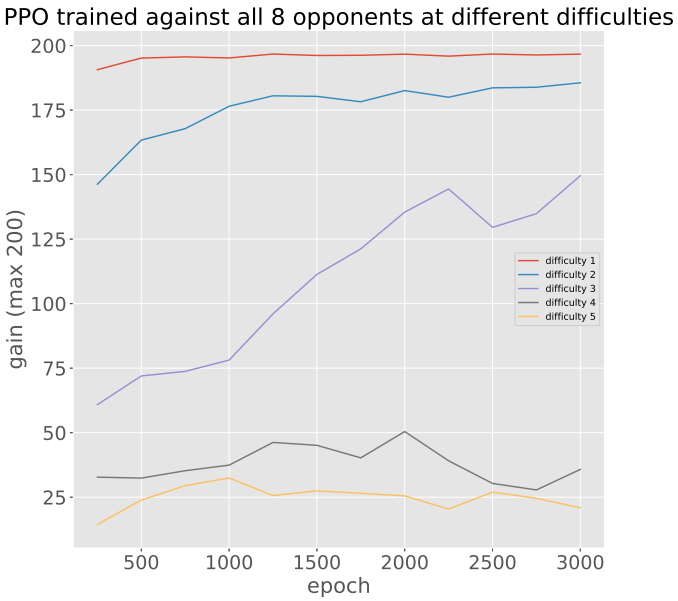

Fig. 7. Gain Evolution for PPO when trained against all 8 opponents. (In case of grayscale viewing: the plots are fully sorted, with difficulty 1 being the highest, and difficulty 5 the lowest.)

Since our generalised PPO model is very close to the previous upper bound, we believe our single-model, all-opponent, all-difficulty exploration should provide a good estimation of what constitutes a good result on all difficulty levels.

We can also point out seemingly more difficult opponents (such as opponent 7), for both gain (Table IV) and loss rate (Table V), and provide more general information on, and understanding of the problem.

TABLE IV
PPO (GENERALISED) 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

TABLE V
PPO (GENERALISED) TRAINED AGAINST ALL OPPONENTS (PERCENTAGE OF 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

VII. CONCLUSIONS

A. Upper Bound

As stated previously, interpreting the initial issue as an exploration versus exploitation trade-off allowed us to repurpose the PPO algorithm. While it wasn't neither exploratory nor flexible enough to provide the best generalisation from 4 opponents to 8, it was exploitative enough to significantly increase the opponent-specific upper bound.

Even with a more modest 1000-epoch runtime, the specialised PPO is able to easily exceed the previous upper bound [3], and approach the maximum gain of 200.01. In fact, it does so for opponent 2, apparently the easiest for our algorithm to defeat, according to Table IV.

This previous upper-bound was established with a specialised (one model per opponent) NEAT algorithm, and we feel the comparison is fair: matching specialised algorithm to specialised algorithm.

While the previous upper bound does not have a time estimate attached to it, given our experience with both methods, we believe our proposed algorithm to be slower than NEAT, due to NEAT's ability to change, and minimse, the structure of the evolved ANN, and its ability to make larger exploratory jumps.

The single-model, all-opponent PPO seems to match the previous upper bound (given the gain range of $[0, 200.01]$, and the range the results take values in, we believe that a difference of 0.1 is not significant); thus, we show that a single model can perform very well against all 8 opponents. How an algorithm might generalise such a model from fewer opponents to all opponents is an open, difficult problem.

We feel this is an important result: while the initial upper bound [3] provided 8 separate models to deal with 8 opponents, we show that there exists a single model which is, in harmonic mean, just as good. We also believe that there is room for improvement for opponent-generalising algorithms. For the first statement, we have proof-by-existence: if our PPO found the model, then it most certainly exists, and is possible to achieve through other algorithms. The second statement - belief - we cannot prove yet; it would require showing that the behaviour patterns learned on one opponent can be applied to others, to a very high degree.

B. Other Difficulty Levels

In both our PSO (Fig. 4) and PPO (Table IV) explorations, we have found the game difficulty to matter a great deal. This is to be expected, as increased difficulty lowers player action effectiveness, and increases opponent effectiveness.

We'd like to propose difficulty 3 as an interesting problem in itself. As shown in Fig. 7, PPO shows the greatest improvement there, from the first to the last epoch. Difficulties 1 and 2 are easier to solve, while 4 and 5 are currently very difficult - but all 4 show relatively flat improvement curves. Algorithms playing Evoman at difficulty 3 could show the greatest variance in results and further guide our algorithm-crafting decisions.

C. Future Work

We believe that, given the difficulty of the generalisation problem, pushing for greater exploration, while retaining our current level of exploitation are both required. We've attempted this with the 2-stage cascade ensemble method, but we believe now that separating the exploratory and exploitative stages was a mistake (guided by our limited computational resources). The game is too difficult for the best strategies to be *local* refinements of simple strategies. In other words, our results show that the function landscape contains multiple wider, shallower strictly-local optima basins, while the attraction basins for global optima are deep and narrow. There is also a degree of optimum dynamics: opponents don't perform the same recipe of actions, but have a limited ability to react to our agent / player.

Retaining the current level of exploration, while increasing exploitation can be done by greatly increasing the computational resources allocated to solving the problem [14]. Based on our results (Fig. 3), we recommend a PSO+PPO Memetic Algorithm [12].

A way to sidestep the increase in computational time could be creating an on-line learning algorithm / agent. Thinking about the problem as a prediction problem - "if half the opponents behaved in their certain ways, how will the rest behave? " - , we can have 2 basic ways of dealing with the prediction.

We can try to predict the full behaviour of the remaining test opponents ahead of time, or we can try to predict each action-by-action, as we confront them. Despite the increase in algorithmic difficulty for on-line learning, the second problem seems far easier than the first. Building on the previous proposal, we'd advise a sparse PSO+PPO Memetic Algorithm. While sparse PPO didn't achieve better results in our exploratory tests, they were evaluated only on the basic strategies.

VIII. Acknowledgments

We'd like to thank the creators and developers of Evoman for their implementation and documentation efforts. We'd also like to thank Ioana-Teodora Norocea for her editing help.

References

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

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

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

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

[5] 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

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

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

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

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

[10] 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

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

[12] Moscato, P. (1989). "On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms". Caltech Concurrent Computation Program (report 826).

[13] Cybenko, G. (1989). "Approximations by superpositions of sigmoidal functions", Mathematics of Control, Signals, and Systems, 2(4), 303–314. doi:10.1007/BF02551274

[14] Wolpert, D.H., Macready, W.G. (1997), "No Free Lunch Theorems for Optimization", IEEE Transactions on Evolutionary Computation 1, 67.

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

[16] Joshua Achiam, 2018, "Spinning Up in Deep Reinforcement Learning" https://github.com/openai/spinningup