

DarkrAI: a Pareto ϵ -greedy policy

Improving Pokémon AI Traning With NSGA-II

Simone Alghisi Samuele Bortolotti Massimo Rizzoli
Erich Robbi

University of Trento

September 1, 2022

Pokémon

Pokémon uses a turn-based system: at the start of each turn, both sides can choose to attack, use an item, switch the Pokémons for another in their party. The Pokemon who strikes first is determined by the Move's Priority and the Pokémons Speed. Each Pokémon uses moves to reduce their opponent's HP until one of them faints, i.e. their HP reach 0. If all of a player's Pokémons faint, the player loses the battle.

Figure 1: Pokémon battle

Reinforcement learning (RL) is an area of Machine Learning where an agent receives a reward based on the action it has performed. Actions allow the agent to transition from a state to another. The final objective is to learn a policy to reach a terminal state with the best reward achievable.

Deep Q-Learning

The reinforcement learning technique we have employed is called *Deep Q-Learning*, which maps input states to a pair of actions and Q-values using an Artificial Neural Network. *Q-Learning* is based on the *Q-function*, namely $Q : S \times A \rightarrow R$, which returns - given a state-action pair $(s, a) \in S \times A$ - the expected discounted reward ($r \in R$) for future states.



NSGA-II is a Evolutionary Algorithm that allows to produce *Pareto-equivalent* (or non-dominated) solutions of a multi-objective optimisation problem.

General idea

The idea is that, given that the search space is very big - there are 10^{354} different ways a Pokémon battle can start, and each turn has at most 306 different outcomes (and only for a single player) - we would like to positively bias our model with a controlled search, removing particularly useless moves, i.e. consider for the most Pareto-equivalent solutions.



Genotype representation

Generally, in a Pokémon battle two actions are possible, i.e. performing a move or a switch. Moreover, depending on the type of battle, it may be necessary to specify the target of the move. To encode such a thing, we came up with the following genotype: each Pokémon is represented using two genes, i.e. action and target (optional) (a, t). The whole genotype tells us who is going to perform what on whom.

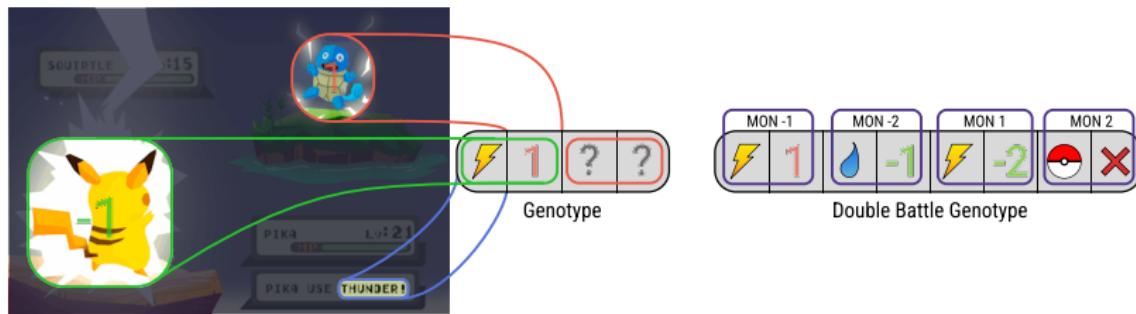


Figure 2: Genotype representation

Genetic operators - Mutation

Mutation is performed for each gene in a genotype with probability $P_m = 10\%$: both the action and the target may be mutated, meaning that it is possible to go from a move to a switch (and vice-versa).

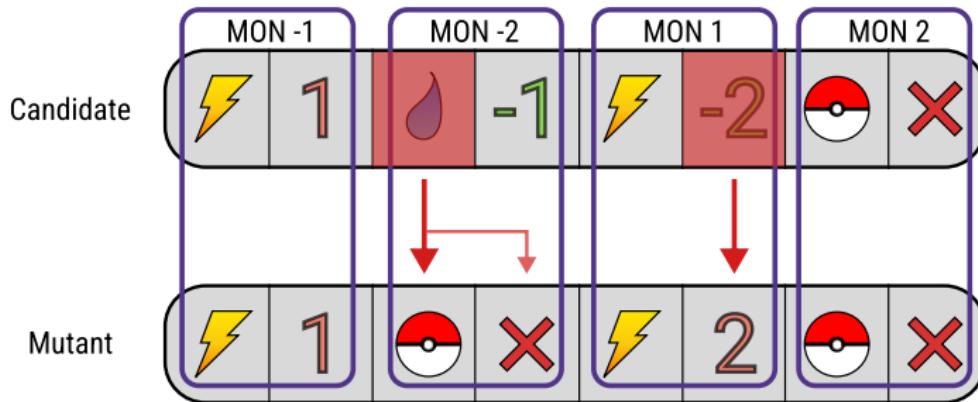


Figure 3: Mutation



Instead, we used Uniform Crossover in a particular way: given that each Pokémon is represented by a valid (a, t) pair, we perform crossover by selecting the whole pair from one of the parents to avoid inconsistencies. Furthermore, crossover is performed with $P_c = 100\%$, and $P_{bias} = 50\%$ (i.e. the bias towards a certain offspring).

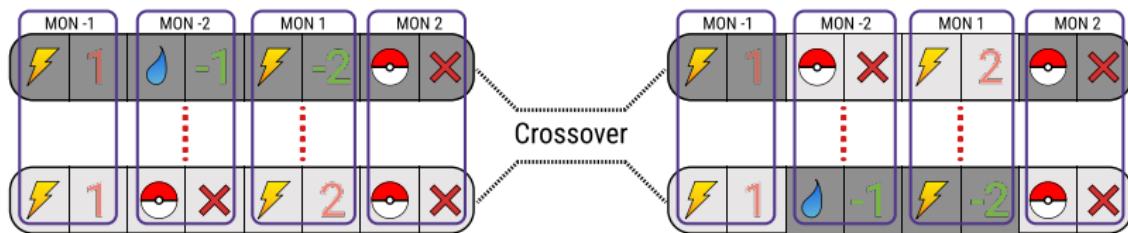


Figure 4: Recombination

Objective & Optimisation

Concerning the *Pareto front* we have considered four variables with the following optimisation problem:

$$\underline{x} = (x_1, x_2, x_3, x_4) \in \mathbb{R}^4 \quad \text{where } \mathbb{R}^4 = \{(x_1, x_2, x_3, x_4) : 0 \leq x_1, x_2, x_3, x_4 \leq 100\}$$

where x_1 is the damage dealt by the ally Pokémons to the opponents, $x_2 is the damage dealt by the opponents' Pokémons to the allies, x_3 is the health points remaining of the player's Pokémons and x_4 is the health points remaining of the opponent's Pokémons.$

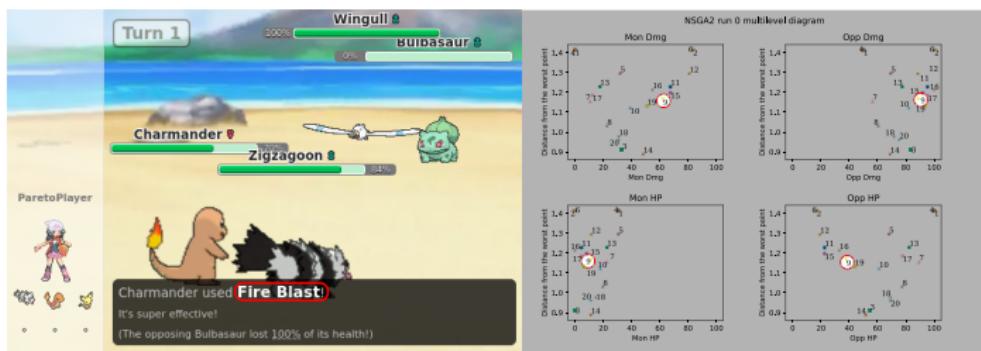


Figure 5: Multi-level diagram plot of the Pareto Front



Architecture details

The agent architecture is a four-layer deep *Multilayer Perceptron (MLP)*, which employs *ReLU* as activation function. In particular:

- input and output layers size depend on the type of battle the network is facing (e.g. a 4 VS 4 battle implies a size of 244 input neurons);
- two hidden hidden layers of size 256 and 128, respectively.

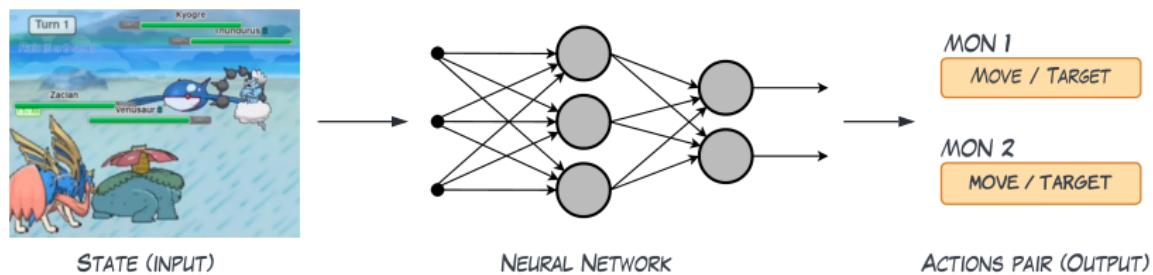


Figure 6: Forward pass of the Artificial Neural Network

The standard agent uses a simple ε -greedy policy:

- it starts from a probability $P_r = 1.0$ to perform a random action;
- it linearly decreases to $P_r = 0.1$ in the first 40% of the training;
- for the remaining 60% of the training it linearly decreases to $P_r = 0.01$.

ParetoPlayer

ParetoPlayer embeds the Pareto search of non-dominated moves: it performs

- a random action using *NSGA-II* with $P_{\text{pareto}} = 0.7$;
- a completely random move with $P_{\text{random}} = 0.3$.

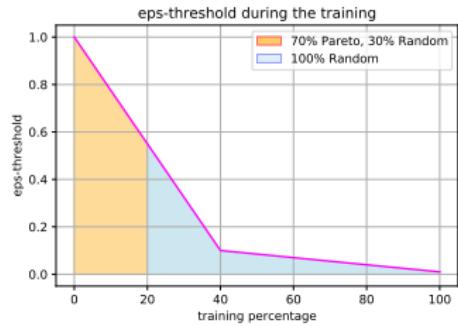


Figure 7: eps-threshold value during the training

Program structure

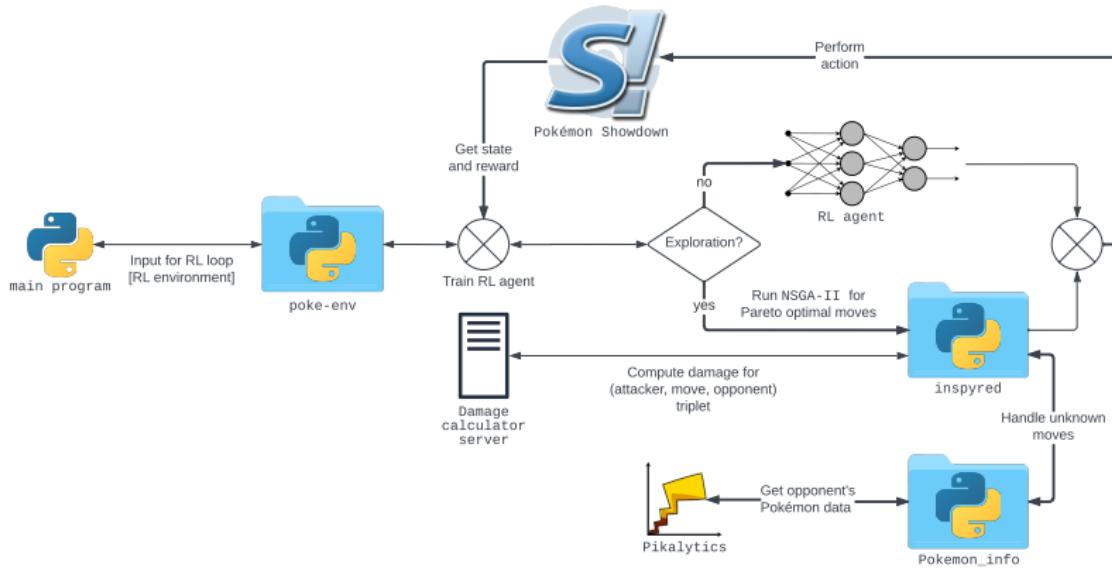


Figure 8: Program structure

Experiments

All agents were trained by having them fight against *MaxDamagePlayer*, i.e. a bot which always chooses the combination of moves that deals the highest amount of damage.

Several situations were considered, such as:

- 2 VS 2 battle with static teams;
- 2 VS 2 battle with the opponent team sampled randomly from a pool of possible Pokémons.



Figure 9: Different battle settings

We have tested both the normality and the statistical significance of the proposed solution with the employment of the following graphical and analytical tools:

Normality

- Quantile-Quantile plot
- Shapiro-Wilk test
- Kolmogorov-Smirnov normality test

Statistical significance

- Box plot
- t-test
- Wilcoxon rank-sum test



Empirical results - Fixed teams

- We expect the episode reward of ParetoPlayer to be higher than the episode reward of Player ($p \leq 2.2 \cdot 10^{-16}$).
- Training runs of ParetoPlayer tends to produce higher reward values ($p \leq 2.886 \cdot 10^{-12}$), but in some cases the rewards are almost equivalent.
- During evaluation ParetoPlayer tends to win more ($p \leq 3.048 \cdot 10^{-5}$)

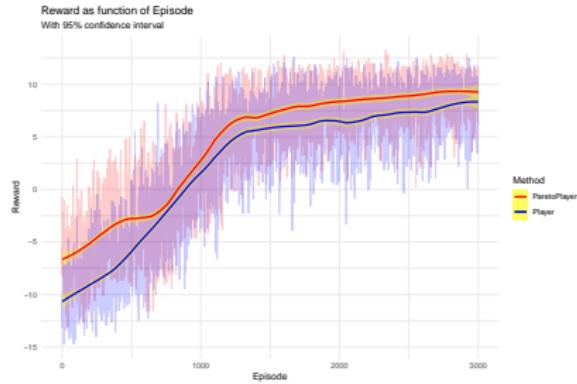


Figure 10: Row-mean reward per episode for Pareto and ParetoPlayer

Empirical results - Fixed Teams

Empirical results - Sampled Teams

- We still expect the episode reward of ParetoPlayer to be higher than the episode reward of Player
- ParetoPlayers' reward distributions have a significant shift location to the right w.r.t to the Player's distribution ($p \leq 0.002278$ and $p \leq 0.01931$)
- The winning percentage is not always in favour of ParetoPlayer (0.716 and 0.673 vs 0.694)

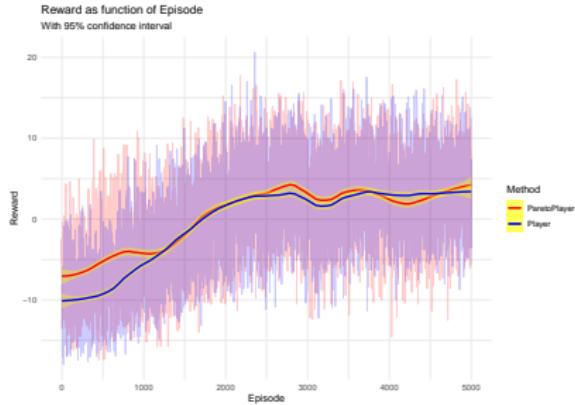


Figure 11: Row-mean reward per episode for Pareto and ParetoPlayer (sampled teams)

The main difficulties we have encountered concern:

- Damage calculator
- Hyperparameters selection and topology search
- Pokémon double battles
- Pokémon battle switches

- ParetoPlayer is able to positively bias the training by providing higher rewards
- when the search space is small enough and a single win condition is presented, Player outperforms ParetoPlayer

Future work

- perform better topology and hyperparameters search
- reduce NSGA-II performance bottleneck (time-consuming operations)
- use another network to properly address forced switch



Thanks for your attention!



Repositories

- pareto-epsilon-greedy-RL
- poke-env (modified)
- Pokemon_info

Collaborators' Github

- Simone Alghisi
- Samuele Bortolotti
- Massimo Rizzoli
- Erich Robbi



Normality - Fixed teams

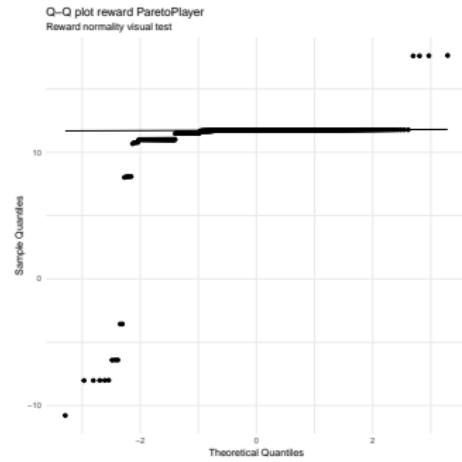


Figure 12: Quantile-Quantile plot episode reward computed on 1000 battles during **ParetoPlayer** model evaluation

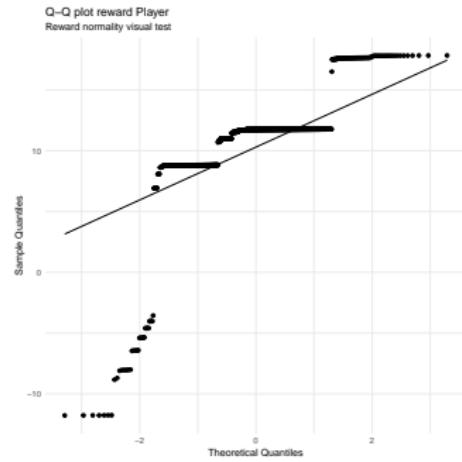


Figure 13: Quantile-Quantile plot episode reward computed on 1000 battles during **Player** model evaluation



Additional results - Box plots

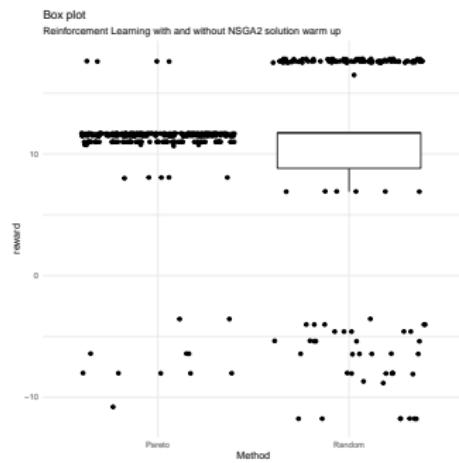


Figure 14: Box plot computed on 1000 battles during ParetoPlayer and Player model evaluation

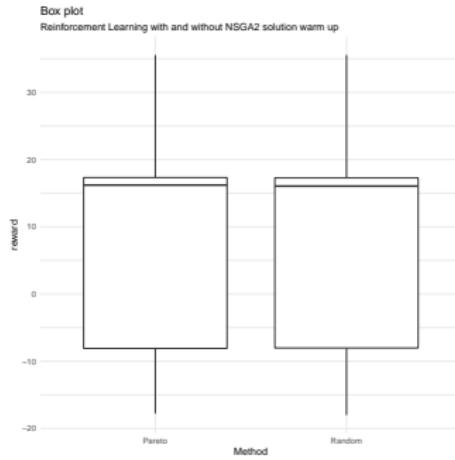


Figure 15: Box plot computed on 1000 battles during ParetoPlayer and Player model evaluation (**with variable enemy team**)

Unknown Moves

At the beginning of the battle the agent does not know which moves the opponent Pokémon have, thus we rely on Pikalytics in order to get the most probable moves in competitive settings. To assign the most probable moves to a Pokémon we:

- ① normalise the probabilities obtained from Pikalytics;
- ② draw a random number;
- ③ sample the corresponding move;
- ④ repeat from 0 until we have a total of 4 moves.



Moves

Iron Head	STEEL	99.985%
Protect	ROCK	99.375%
Sacred Sword	FIGHTING	75.735%
Play Rough	PURE	53.830%
Quick Attack	ROCK	28.244%

Figure 16: Zacian's possible moves on Pikalytics

Among all the possible information, we focused on the following:

- the percentage of Pokémons alive;
- the weather;
- the field condition;
- for each Pokémon we considered:
 - type (e.g. fire, grass, ect.);
 - HP percentage;
 - statistics (normalised);
 - status (e.g. asleep, poisoned, ect.);
 - for each of its move, we considered: id, priority, type, and damage it deals to the opponent active Pokémons.



Fitness Evaluation

In order to get a good fitness evaluation of our turn, we perform the following:

- ① analyse the previous turn;
- ② estimate the statistics of the opponent;
- ③ predict a possible turn order based on the Moves Priority and Pokémons' Speed;
- ④ prepare the field by handling switches;
- ⑤ compute the damage by either:
 - sending a request to the server;
 - retrieve a previous result.

252+ Atk Abomasnow Wood Hammer vs. 0 HP / 0 Def Abomasnow: 108-127 (33.6 - 39.5%) -- guaranteed 3HKO

Possible damage amounts: (108, 109, 110, 111, 113, 114, 115, 117, 118, 119, 120, 122, 123, 124, 126, 127)

Pokémon 1

Type: Grass, Ice
Forme: Abomasnow
Gender: Male
Level: 100

	Base	IVs	EVs
HP	90	31	0 0 321
Attack	92	31	252 0 311
Defense	75	31	0 0 186
Sp. Atk	92	31	0 0 198
Sp. Def	85	31	4 0 207
Speed	60	31	252 0 219

Pokémon 2

Type: Grass, Ice
Forme: Abomasnow
Gender: Male
Level: 100

	Base	IVs	EVs
HP	90	31	0 0 321
Attack	92	31	252 0 311
Defense	75	31	0 0 186
Sp. Atk	92	31	0 0 198
Sp. Def	85	31	4 0 207
Speed	60	31	252 0 219

Figure 17: Damage calculator request



UNIVERSITÀ
DI TRENTO

Previous Turn Analysis

To have a better estimate of the next possible turns, the previous turn is analysed to extract unknown information (e.g. Pokemon's stats, item, etc.). In particular we

- ➊ retrieve the previous turn;
- ➋ extract the actions performed;
- ➌ simulate the actions execution using our current knowledge;
- ➍ compare the actual turn with the one estimated;
- ➎ adjust our beliefs if needed.

Stats

Base stats

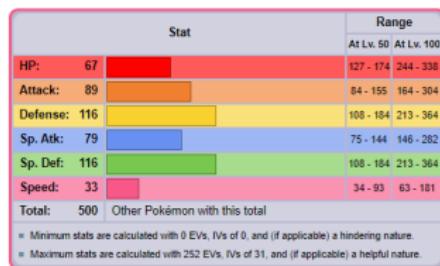


Figure 18: A Pokémon's base stats

The Shapiro-Wilk test is a test of normality which is frequent in statistics and it is based on the expected values of the order statistics. Its null hypothesis is whether a sample $\{x_1, \dots, x_n\}$ came from a **normally distributed population**:

- Thus, if the p -value is less than the chosen α level, then the null hypothesis is rejected and there is evidence that the data tested are not normally distributed;
- On the other hand, if the p -value is greater than the chosen α level, then the null hypothesis (that the data came from a normally distributed population) can not be rejected.

In statistics, the Kolmogorov-Smirnov test is a nonparametric test of the equality between two distribution, namely it does not assume anything about the underlying data distribution. Its null hypothesis is whether the two set of samples were drawn from the **same probability distribution**:

- Thus, if the p -value is less than the chosen α level, then the null hypothesis is rejected and there is evidence that the data tested are not drawn from the same distribution, namely one group stochastically dominates the other;
- On the other hand, if the p -value is greater than the chosen α level, then the null hypothesis (that the data came from the same distribution) can not be rejected.



The Wilcoxon rank-sum test is one of the most powerful non-parametric test which is used to compare two groups of continuous measures. Its null hypothesis is whether the two populations have the **same distribution and the same median**:

- Thus, if the p -value is less than the chosen α level, then the null hypothesis is rejected and there is evidence that the data tested are not drawn from the same distribution, namely one distribution is shifted to the left or right of the other;
- On the other hand, if the p -value is greater than the chosen α level, then the null hypothesis (that the data came from the same distribution) can not be rejected.



Empirical results - Sampled teams [First Game]

Empirical results - Sampled teams [Second Game]