Raport: Algorytmy Ewolucyjne

Tymoteusz Urban 320665

14 czerwca 2024

1 Wstęp

W niniejszym raporcie przedstawione zostały wyniki i analiza modelu algorytmu genetycznego. Algorytm został zaimplementowany od zera, bez użycia gotowych bibliotek implementujących rozwiązania typu DL i ML. Celem było zrozumienie działania algorytmów ewolucyjnych i wpływu parametrów na proces ewolucji.

2 AE 1

Pierwsze zadanie polegało na napisaniu podstawowego algorytmu genetycznego z mutacją gaussowską i krzyżowaniem jednopunktowym, oraz sprawdzeniu go na dwóch podanych funkcjach. Zastosowana została selekcja ruletkowa.

2.1 Funkcja 3-wymiarowa

Dana trójwymiarowa funkcja była opisana wzorem:

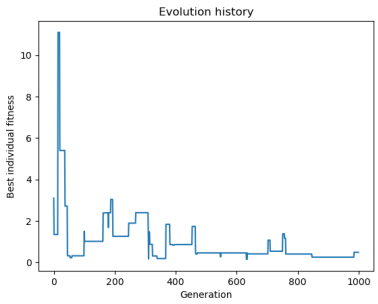
$$f(x) = x^2 + y^2 + 2z^2$$

Przeprowadzone zostało 5 eksperymentów z różnymi parametrami rozmiaru populacji i liczby generacji dla ustalonego łącznego kosztu $rozmiar * generacje = 5000$. *Mutation rate* wynosił 0.2, a *crossover rate* był równy 0.7:

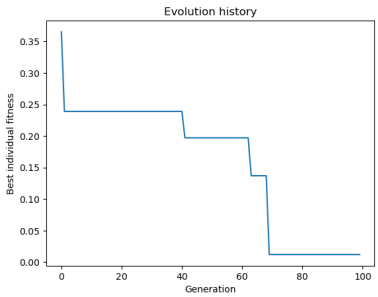
populacja	liczba generacji	końcowy wynik
5	1000	0.4841
10	500	0.0424
50	100	0.0121
100	50	0.0905
500	10	0.0171

Tabela 1: Wyniki

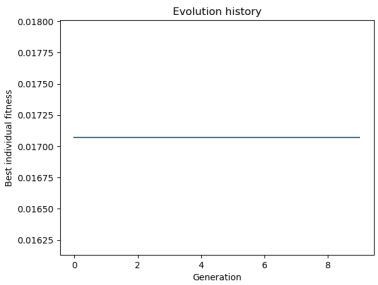
Widzimy, że najlepszy wynik został osiągnięty dla populacji 50 i liczby generacji równej 100. Intuicyjnie też wydaje się to najrozsądniejszy podział. Bardzo dobry wynik został także osiągnięty dla populacji 500 i liczby generacji 10, ale na wykresie poniżej możemy zobaczyć, że wynikało to z prostoty zbioru danych, gdyż już w pierwszej iteracji udało się wylosować bardzo dobrego osobnika. Przy bardziej złożonej funkcji kosztu nie byłoby to możliwe.



(a) P=5, G=1000



(b) P=50, G=100



(c) P=500, G=10

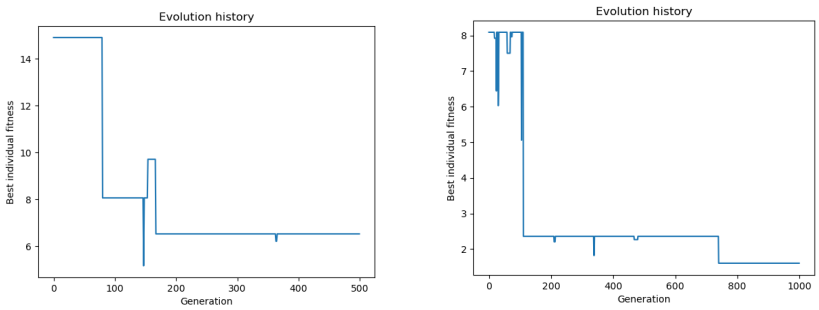
Rysunek 1: Historia ewolucji

2.2 Funkcja Rastrigina

Drugą funkcją do przetestowania była 5-wymiarowa funkcja Rastrigina z globalnym minimum w 0. Jest powszechnie używana do testowania algorytmów optymalizacyjnych, gdyż posiada bardzo wiele minimów lokalnych. Oto wyniki:

populacja	liczba generacji	końcowy wynik
10	500	6.529
50	100	8.057
100	1000	1.604

Tabela 2: Wyniki



(a) P=10, G=500 (b) P=100, G=1000

Rysunek 2: Historia ewolucji

Możemy zauważyć, że oczywiście im większa populacja i im więcej generacji, tym lepsze wyniki. Niestety nasz algorytm wpada w minima lokalne i nie chce z nich wyjść. Nie pomaga także fakt, że czsami najlepsze osobniki wypadają z populacji. W następnych zadaniach zostało to poprawione i przed selekcją ruletkową 10% najlepszych osobników jest zachowywanych.

3 AE2

Kolejnym zadaniem było stworzenie rozwiązania dla *cutting stock problem* poprzez algorytm genetyczny. Początkowa koncepcja na rozwiązanie problemu wyglądała następująco:

- **Reprezentacja problemu**: każdy osobnik to lista prostokątów w kole, gdzie każdy prostokąt jest reprezentowany przez pozycję lewego dolnego rogu, wymiary prostokąta i wartość.
- **Funkcja fitness**: liczymy sumę wartości wszystkich prostokątów w rozwiązaniu.
- **Inicjalizacja**: najpierw wstawiamy deski o największej wartości na jednostkę powierzchni, a następnie upychamy kolejne o coraz mniejszej wartości. Wstawiamy losowo bądź algorytmicznie od góry od lewej
- **Mutacja**: próbujemy przesuwać lewo/prawo i góra/dół (bez odwracania) i dołożyć jakieś prostokąty.
- **Krzyżowanie**: jednopunktowe, losujemy indeks na osi y, w tym miejscu przecinamy dwóch rodziców. Z jednego bierzemy prostokąty nachodzące na ten indeks i wszystkie wyżej, a z drugiego nachodzące plus wszystkie niżej. Potem usuwamy te prostokąty, które na siebie nachodzą. Ewentualnie identyczność.
- **Selekcja**: ruletkowa

Ostatecznie okazało się, że zwykły deterministyczny algorytm radzi sobie świetnie i spełnia wszystkie progi punktowe z dużą nadwyżką. W związku z tym koncepcja została nieco zmieniona, a deterministyczny algorytm zmodyfikwany na algorytm genetyczny:

- **Reprezentacja problemu**: jak wyżej
- **Funkcja fitness**: jak wyżej
- **Inicjalizacja**: Dla każdego rodzaju deski, poczynając od deski o największej wartości na jednostkę powierzchni, wykonujemy w pętli następujące kroki:

1. losujemy punkt w kwadracie opisanym na kole (tak naprawdę to z prostokąta: z wymiarów kwadratu od góry i od prawej strony odejmujemy odpowiednio wysokość i szerokość deski, gdyż wtedy w tamtym obszarze deska na pewno wychodzi poza obręb koła)
2. sprawdzamy czy deska w tym punkcie nie wychodzi poza obręb koła ani czy nie nachodzi na inne deski
3. jeśli oba warunki są spełnione, dodajemy deskę do rozwiązania, wpp. zwiększamy liczbę nieudanych prób o 1
4. jeśli liczba nieudanych prób przekracza dany threshold (np. 20), przechodzimy do kolejnej deski

- **Mutacja**: Próbujemy przesunąć każdą deskę do góry i następnie dołożyć jakieś prostokąty w sposób analogiczny do inicjalizacji. Proces przesuwania wygląda następująco:

  1. Znajdź wszystkie prostokąty na pionowej obszarze wyznaczonym przez szerokość przesuwanej deski.
  2. Oblicz pionową przestrzeń między każdym znalezionym prostokątem.
  3. Jeśli istnieje pionowa przestrzeń większa niż wysokość deski, umieść tam deskę.
  4. Jeśli nie ma takiej przestrzeni, umieść ją pod najniższym z prostokątów (jeśli nie wychodzi poza okręg).
  5. Jeśli w pionowym obszarze nie został znaleziony żaden prostokąt, umieść deskę w najwyższym możliwym miejscu na wyznaczonym obszarze.

  Mutowany był każdy osobnik, a osobniki zmutowane zastępowały swoich rodziców.

- **Krzyżowanie**: brak
- **Selekcja**: brak

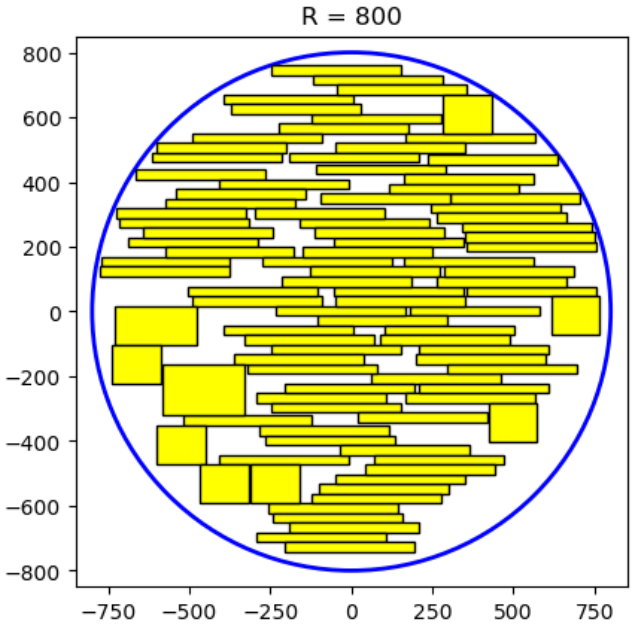
3.1 R = 800

Pierwszym zbiorem był zbiór z prostokątami, które trzeba było umieszczać w kole o średnicy 800. Algorytm został odpalony z następującymi parametrami:

Parametr	Wartość
rozmiar populacji	20
liczba generacji	30
liczba prób	150

Tabela 3: Parametry

Pozwoliło to na osiągnięcie progu punktowego już w 3 iteracji (fitness@1gen =22440, fitness@3gen=30340). Końcowa wartość prostokątów wyniosła 35380.



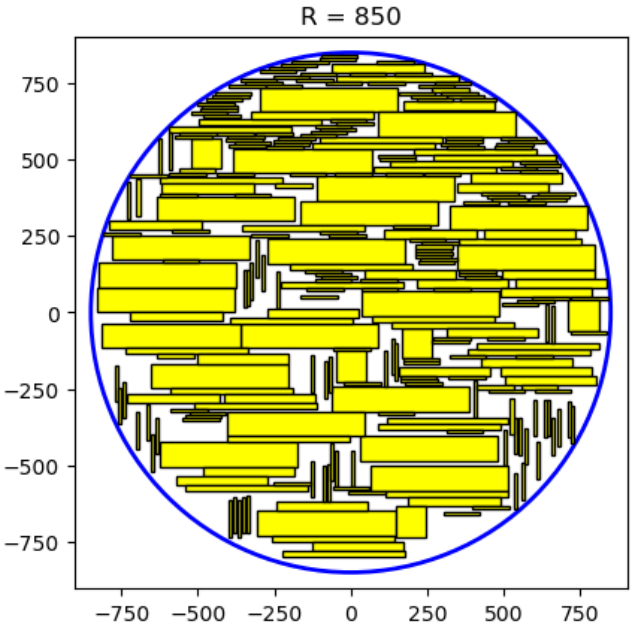
Rysunek 3: Wizualizacja najlepszego osobnika

3.2 R = 850

Dla tej wartości średnicy nie trzeba było spełniać progów punktowych.

Parametr	Wartość
rozmiar populacji	10
liczba generacji	20
liczba prób	120

Tabela 4: Parametry



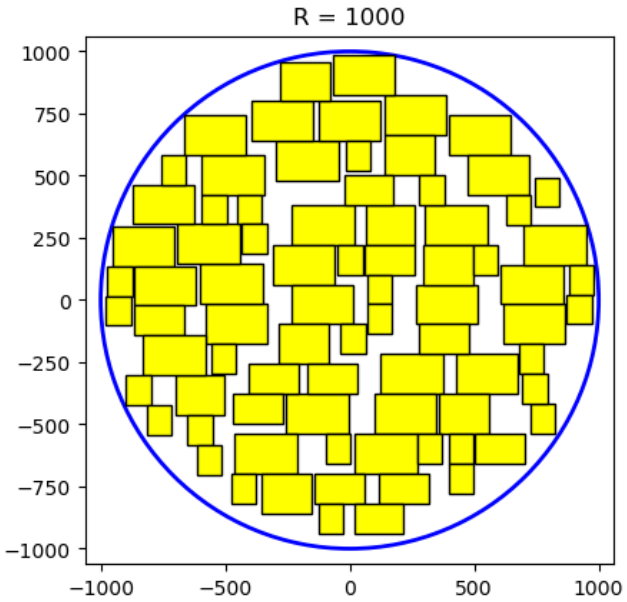
Rysunek 4: Wizualizacja najlepszego osobnika

3.3 R = 1000

Dla zbioru z średnicą 1000 próg punktowy wynosił 17500. Osiągnięty wynik wynosił 20840, a próg został osiągnięty już w 2 iteracji.

Parametr	Wartość
rozmiar populacji	10
liczba generacji	20
liczba prób	120

Tabela 5: Parametry



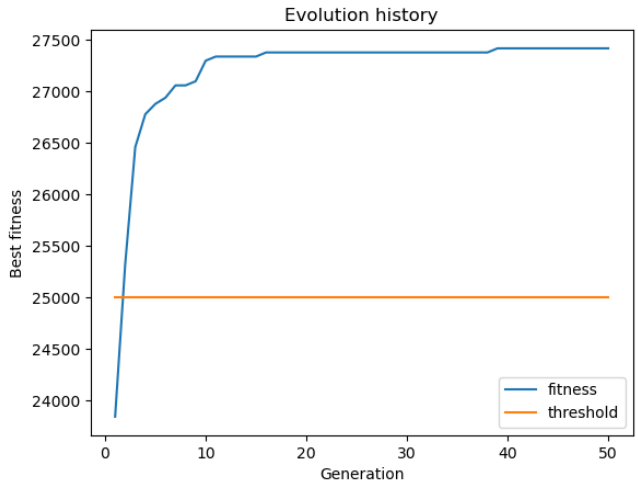
Rysunek 5: Wizualizacja najlepszego osobnika

3.4 R = 1100

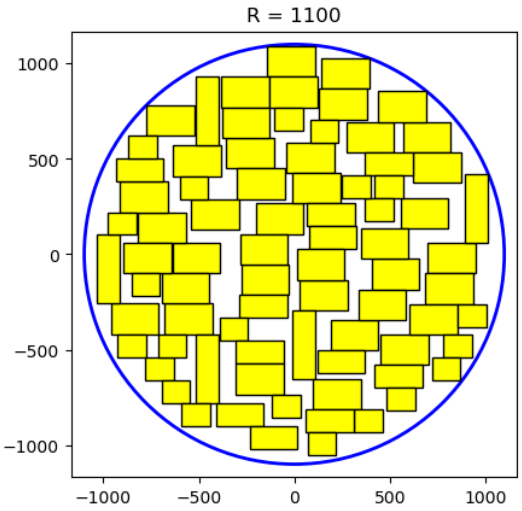
Dla zbioru z średnicą 1100 próg punktowy wynosił 25000. Osiągnięty wynik wynosił 27480, i tutaj próg także został osiągnięty już w 2 iteracji.

Parametr	Wartość
rozmiar populacji	20
liczba generacji	50
liczba prób	200

Tabela 6: Parametry



(a) Historia generacji



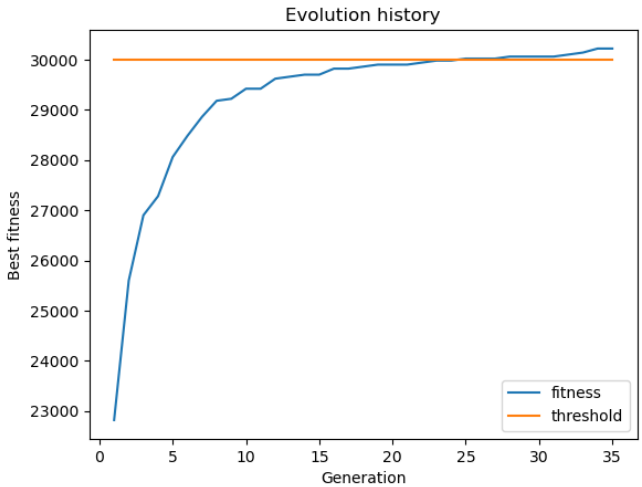
(b) Wizualizacja najlepszego osobnika

3.5 R = 1200

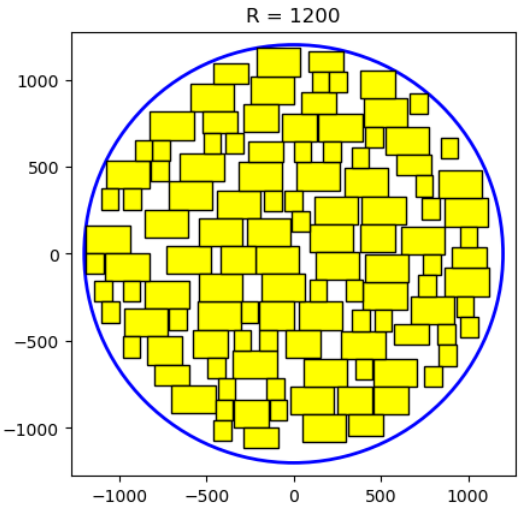
Zbiór ze średnicą 1200 był jedynym przypadkiem, gdzie osiągnięcie progu punktowego nie było łatwe. Próg został osiągnięty dopiero po 25 generacjach, a przy wielu próbach nie był w ogóle osiągany.

Parametr	Wartość
rozmiar populacji	20
liczba generacji	35
liczba prób	120

Tabela 7: Parametry



(c) Historia generacji



(d) Wizualizacja najlepszego osobnika

Po przeprowadznych testach widzimy, że największy wpływ na dobre wyniki ma inicjalizacja i pierwsze generacje. Jedynym wyjątkiem była średnia 1200, gdzie algorytm potrzebował czasu i wielu prób żeby osiągnąć próg punktowy. Możemy wyciągnąć wnioski, że rzucanie się na ten problem z algorytmem genetycznym nie ma za dużo sensu. Zwykły algorytm, który wstawiając deski od razu przesuwa je maksymalnie do góry (czyli przeprowadzana jest tylko zoptymalizowana inicjalizacja), o wiele szybciej osiąga o wiele lepsze wyniki. Tworzony i modyfikowany jest tylko 1 osobnik a nie 20 osobników modyfikowanych 35 razy. Dla porównania:

algorytm	rozmiar populacji	liczba generacji	wartość
Ewolucyjny	20	50	27480
Zwykły	1	1	36500

Tabela 8: Zbiór R = 1100

algorytm	rozmiar populacji	liczba generacji	wartość
Ewolucyjny	20	35	30220
Zwykły	1	1	38580

Tabela 9: Zbiór R = 1200

4 AE3

Ostatnim zadaniem było zaimplementowanie uczenia sieci MLP z użyciem algorytmu genetycznego. Pojedynczym osobnikiem jest instancja klasy MLP, a przy mutacjach i krzyżowaniu zmianom poddawane są wagi i biasy. Najpierw wszystkie wektory wag i biasów są transformowane do jednowymiarowej listy, następnie poddawane mutacji lub krzyżowaniu i ponownie przekształcane na odpowiednie wektory wag i biasów. Krzyżowanie jest klasyczne jednopunktowe, a mutacja gaussowska przebiega w następujący sposób:

- stworzenie 1-wymiarowego wektora z wag i biasów
- standaryzacja wektora
- wylosowanie liczby mówiącej ile indeksów będzie modyfikowanych
- wylosowanie indeksów do zmiany
- dodanie losowych wartości z rozkładu standardowego
- destandaryzacja
- ponownie przekstrzałcane na odpowiednie wektory wag i biasów

Następnie 10% najlepszych osobników jest zachowywanych, a na pozostałej części nowej populacji aplikowana jest selekcja ruletkowa. Wszystkie eksperymenty zostały przeprowadzone z $mutationrate = 0.7$ i $crossoverrate = 0.7$, chyba że zostało stwierdzone inaczej.

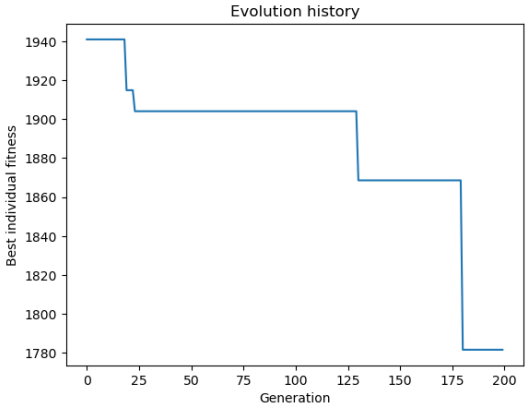
4.1 Multimodal Large

Pierwszy zbiorem do sprawdzenia był zbiór *multimodal large*, który przeznaczony jest do zadania regresji. Testowana architektura składała się z warstw sigmoidalnych oprócz ostatniej, która była liniowa. Początkowe sprawdzone zostały 3 architektury, przy czym architektura z warstwami $[1, 10, 10, 1]$ liczyła się długo i dawała słabe efekty, więc nie była dalej testowana.

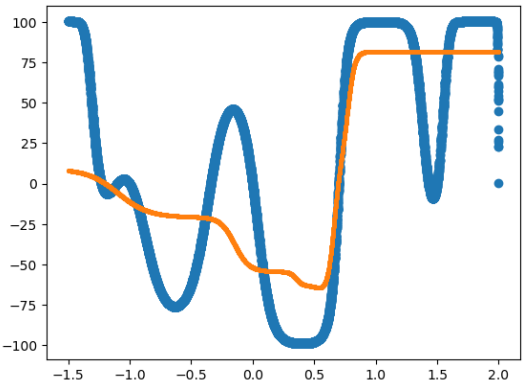
architektura	rozmiar populacji	liczba generacji	średnie MSE	odchylenie std MSE
$[1, 10, 1]$	30	150	2175	385
$[1, 5, 5, 1]$	30	150	2487	274

Tabela 10: Wyniki eksperymentów, 5 prób na architekturę

Pojedyncza warstwa z 10 neuronami sprawuje się o wiele lepiej. Potrafiła osiągnąć MSE mniejsze od 1700, podczas gdy architektura $[1, 5, 5, 1]$ nigdy nie zeszła poniżej MSE 2150. Jednakże tak ogólnie algorytm wypadł bardzo słabo. Przy klasycznym treningu MLP z momentem z łatwością można było osiągnąć MSE mniejsze niż 10 na tym zbiorze.



(e) Historia generacji

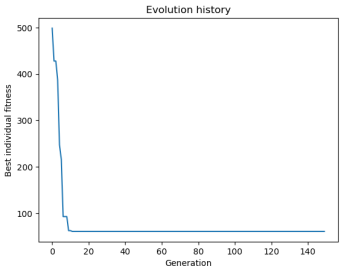


(f) Porównanie predykcji ze zbiorem testowym

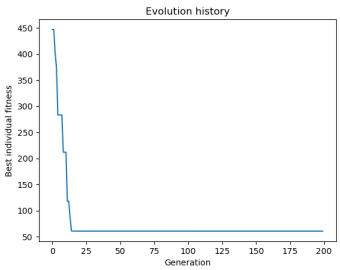
Rysunek 6: Przykładowy trening dla architektury $[1, 10, 1]$

4.2 MPG

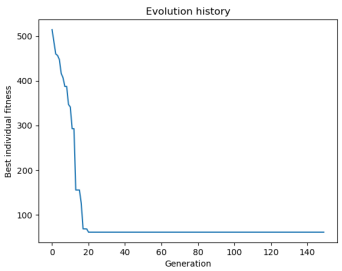
Kolejnym zbiorem był zbiór auto-mpg, także do zadania regesji. Wykonałem na nim kilka eksperymentów z róznym układem warstw i za każdym razem trening zatrzymywał się na MSE równym 60 i nie chciał zejść niżej.



(a) 7,5,1



(b) 7,10,1



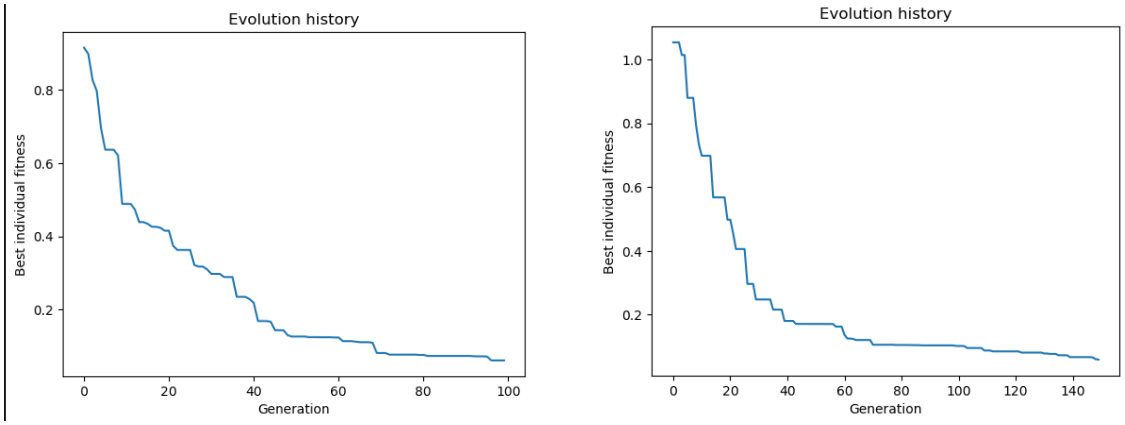
(c) 7,5,5,1

Rysunek 7: Historia ewolucji dla różnych architektur

Po dokładniejszym zbadaniu wynikowych sieci okazało się, że dla każdego modelu samochodu predykcja jest taka sama i wynosi 23.44899391. Wektory wag i biasów wyglądały jednak normalnie, nie było takiej sytacji że wagi się wyzerowały i został tylko ostatni bias. Dodatkowo wpływ na ostateczną predykcję miały tylko tylko 2 i 3 kolumna (cylinders i horsepower). Gdy ustawiało się je na zero, wynik się zmieniał, ale dla typowego zakresu tych wartości jest zwracane zawsze to samo. Pierwszą próbą naprawy była standaryzacja wszystkich danych, a także zwiększenie *mutation rate* i *crossover rate* do 0.9. Niestety, żaden z tych pomysłów nie zmienił sytuacji. Następnie zmodyfikowałem selekcję i zwiększyłem *mutation rate* do 1 licząc na większe skoki i przekroczenie magicznej bariery MSE 60. Niestety, to tylko pogorszyło sprawę. Modyfikacja mutacji tak, by wszystkie elementy wektora były mutowane także nie naprawiła sprawy. Niestety czegokolwiek bym nie próbował, wszystko algorytmy ewolucyjne niezależnie od parametrów zbiegały do tej jednej wartości.

4.3 Iris

Ostatnim zbiorem był zbiór Iris, ale tym razem należało dokonać klasyfikacji. Był to jedyny zbiór, na którym algorytm ewolucyjny spisał się bardzo dobrze. Wytestowałem prostą architekturę z jedną warstwą ukrytą z 5 neuronami. Algorytm był zbieżny i z każdą iteracją krosentropia malała.



(a) 100 iteracji

(b) 150 iteracji

Rysunek 8: Przykładowe treningi dla architektury $[4, 5, 3]$ i populacji 30

Zbadałem także *f1 score* i po agregacji wyników z 5 eksperymentów średni *f1 score* był równy 0.97 a jego odchylenie standardowe 0.01. Widzimy że jest to świetny wynik i naprawdę trudny byłoby go poprawić.

4.4 Podsumowanie

Trenowanie sieci MLP przy pomocy algorytmu ewolucyjnego było na pewno prostsze, niż pisanie obliczeń gradientów i klasycznej aktualizacji wag. Niestety w większości przypadków się nie sprawdziło. Na prostym zbiorze jakim jest Iris algorytm był wystarczający, jednak dla jakiegokolwiek bardziej złożonego problemu nalezy korzystać z klasycznych rozwiązań.